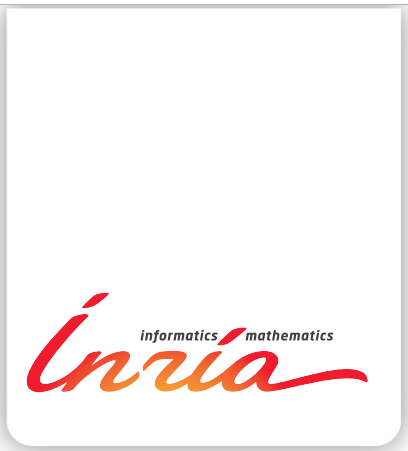
Models of Hippocampus for pavlovian learning

Román Gorojovsky, Frederic Alexandre

RESEARCH REPORT
N° 8377
Octobre 2013
Project-Teams MNEMOSYNE

ISSN 0249-6399 ISRN INRIA/RR--8377--FR+ENG



Models of Hippocampus for pavlovian learning

Román Gorojovsky*, Frederic Alexandre†

Project-Teams MNEMOSYNE

Research Report n° 8377 — Octobre 2013 — 77 pages

Abstract: Amygdala and hippocampus are key structures in pavlovian conditioning. From existing models of these cerebral structures, we test them on corpus, to assess their capacities and also their performances in critical situation. From these results, we propose adaptations to these models together with an early study on forgetting.

Key-words: hippocampus, pavlovian learning, amygdala, learning, cortex

* Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires.
† INRIA Bordeaux – frederic.alexandre@inria.fr

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Modèles de l'hippocampe pour le conditionnement pavlovien

Résumé : L'amygdale et l'hippocampe sont des structures clé en conditionnement pavlovien. A partir de modèles existants de ces structures, nous les testons sur des corpus, afin d'évaluer leurs capacits et leurs performances dans des situations critiques. A partir de ces résultats, nous proposons des adaptations de ces modèles, ainsi qu'une étude préliminaire sur l'oubli.

Mots-clés : hippocampe, aprentissage pavlovien, amygdale, apprentissage, cortex

1 Introduction

This text documents the ideas developed during an internship for the team[1], their implementations, tests and results. The first section describes the model as it existed when this work started, and the basic names and ideas used all along the document[2]. The following sections present the different ideas we worked on, the scenarios used to test them, present the results and discuss them.

The research documented here is divided on three main areas: **Normal working mode** of the model, **Forgetting on the Hippocampus** and the **Splitted Amygdala**. We define and explain these ideas on the sections 3 to 5. The next section describes the scenarios used to test each of these three features, and the results of those tests, if any, appear on the following section. A discussion of those results (or lack thereof) is on the following section.

Finally we discuss what remains open for further work and document some ideas and intuitions on the different problems that are "on the air" in different lines of investigation of the group and how this work and its components relate to them.

After that, on a series of appendices, we mention a few lateral developments from these days that aren't particularly relevant to any of the main ideas and document the tools, scripts and configuration files used during these months. All the code can be found on the forge, on the group's repository.

A note on nomenclature: To differentiate between the biological subsections of the brain and the modules on our computational model, we'll be using the following convention: *italicized* words refer to the biological ones, while `monospaced` words refer to the module/s.

[1] Between april and october 2013, under the INRIA program of international internships
[2] Including this introduction, bit of a circular problem here

Contents

1 Introduction **3**

2 Starting model **7**

2.1 Objective and general ideas . . . 7

2.2 The model . . . 7

2.2.1 `Hippocampus` . . . 7

2.3 The brains . . . 8

2.4 Base scenario, tests and results . . . 8

2.4.1 The Simple XOR scenario . . . 8

2.4.2 Results . . . 9

3 Normal working mode **12**

3.1 Introduction . . . 12

3.1.1 Capacity . . . 12

3.1.2 Scaling . . . 12

3.1.3 Learning by heart . . . 12

3.2 Testing of capacity and scaling . . . 12

3.2.1 Results . . . 13

3.2.2 Discussion . . . 15

3.3 Testing for Learning by heart . . . 16

3.3.1 Results . . . 18

3.3.2 Discussion . . . 34

4 Forgetting on the Hippocampus **39**

4.1 Detailed model of `CA3` . . . 39

4.2 Mechanics of forgetting . . . 40

4.2.1 Global (systematic) forgetting . . . 40

4.2.2 Dynamic forgetting . . . 41

4.3 Modifications to the recall mode . . . 42

4.4 Testing and results . . . 42

4.5 Discussion . . . 49

5 Splitted Amygdala **50**
5.1 Introduction . . . 50
5.2 Architecture . . . 50
5.2.1 Deciders . . . 50
5.3 Implementation details . . . 51
5.3.1 Fine tuning . . . 52
5.4 Testing scenarios . . . 52
5.5 Results . . . 54
5.5.1 Simple XOR . . . 54
5.5.2 Double map XOR . . . 56
5.6 Discussion . . . 65
5.6.1 Simple XOR . . . 65
5.6.2 Double Map XOR . . . 65
5.6.3 Problems with the `Amygdala` . . . 65

6 Further work **68**
6.1 Dynamic forgetting . . . 68
6.2 Consolidation . . . 68
6.3 Decider . . . 68
6.4 Binary vs. Real Computation . . . 69
6.5 Continuous use . . . 69
6.6 `CA1`, the weak link . . . 70
6.7 Probes and better test tools . . . 70

A Other implementations **71**
A.1 Corpus . . . 71
A.2 Random Recall . . . 71

B `main_bis` **72**

C Configuration files for the Scenarios **73**

D Scripts **74**
D.1 Requirements . . . 74
D.2 `plot_amygdala.sh` . . . 74
D.3 Plotting scripts . . . 74

E Unit tests **76**

2 Starting model

2.1 Objective and general ideas

The objective of our modeled brain is to associate unconditional stimuli to neutral stimuli predicting them (pavlovian conditioning). Given some non-learned stimuli or *CS*, which can be external or internal it learns to predict an outcome from them. We call these outcomes *US*.

We have three components for the learning: the *Amygdala*, the *Cortex* and the *Hippocampus*. The first one is a simple neuronal network that takes an input and learns to build a prediction. It also sends an error signal that can be used to modulate the learning and/or unlearning of the other two components.

The *Cortex* one slowly learns associations of these elements to extract minimal cues describing them, while the *Hippocampus* one learns by heart some associations, and is able to recall the recorded ones with only a part of them, which is particularly useful to quickly get the *US* given an input.

2.2 The model

This model is better described on [2] and [1], two publications by its developers. The focus of this work was on the `Hippocampus`, so we'll only give a brief description of the `Amygdala` and the `Cortex`, and refer to those publications for details. For this same reason, we'll only present very broad intuitions of the whys behind some of our results.

The `Amygdala` is a simple neural network with one input level of as many neurons as bits on the inputs, all connected to the output level, which has as many neurons as bits on the output or prediction. The network learns by adjusting its weights to adjust the prediction to the US, once it arrives. The `Cortex` has the same sensory input layer as the amygdala and several additional layers that can be used to extract, by learning, conjunction of activities in layers below, as a model of semantic memory.

2.2.1 `Hippocampus`

Modes of working We map some of the functionalities of the *Hippocampus* to what we call *modes* of work of the `Hippocampus`. These modes aren't mutually exclusive, and part of the work is deciding when each mode is to be used. The modes implemented are **Learn** and **Recall**.

In both modes `Hippocampus` receives an input, which comes from the `Cortex` to be processed. On learning mode, the `Hippocampus` will try to store it, regardless of whether it has already seen it or not. Learning mode is usually enabled after the US has arrived, and that will be part of the stimulus stored. This is important because in recall mode, if the input is already stored, the component will try to *complete* it with its US. In the current implementation, recall mode is always enabled.

The `Hippocampus` has to decide whether the stimulus is new or if it already knows it. This is the problem of *separation and completion*: if two semantically different inputs are syntactically too close, the second to arrive will be considered the same as the first and it will be completed to the first one, with its US.

Architecture The `Hippocampus` is a set of components, each of which represents and models a different part or area of the *Hippocampus*. These parts are, in order of "data flow":

`Entorhinal Cortex` Acts as an entry point/interface with the `Cortex`. Right now is just a pair of buffers, one for the input from `Cortex` and another for the output. The input is the `Cortex` activity plus the US.

Dentate Gyrus or `DG` transforms the input received by the `Entorhinal Cortex` to the input used by the following module, `CA3`. This transformation should make the input sparser.

The `DG`'s output is wider than its input. For an input of size n_i the output's size is $n_o = \frac{n_i^2 - n_i}{2}$, i.e. the number of possible pairs of elements of the vector. Each cell of the output converges to the product of two cells of the input. In some models, the original input is also appended to `DG`'s output, but that's not the case in our implementation.

`CA3` **(Cornu Armonis 3)** This is the actual data storage, the module that actually learns and remembers data. As said before, it has it's own "language", i.e. the data stored is represented differently than in the rest of the model. The details of the implementation of this module can be seen in the section 4.1.

`CA1` Is the module that receives the output from `CA3`, translates it back to the format the rest of the model understands, and sends this to the `Entorhinal Cortex`. Unlike the translation on `DG` this one is learned using an array of perceptrons.

2.3 The brains

There are three different "brains" implemented to test different things at various moments of the development of the model:

- `BrainCA` Only has a `Cortex`, which takes the sensorial input and an `Amygdala` for the final prediction.
- `BrainHA` Only has a `Hippocampus`, which also takes only the CS as input and an `Amygdala`.
- `BrainCHA` Has both `Cortex` and `Hippocampus`, which here takes all the `Cortex`'s activity as input, plus the `Amygdala`

2.4 Base scenario, tests and results

2.4.1 The Simple XOR scenario

All of the scenarios used to test the different features of the system are modifications of this one that we used as a baseline, as something used for comparison. Is a simple non linear problem with two possible inputs, A and B that, on their own, are associated with a positive US, and when presented together (AB) with a negative one. Although two bits on the input are enough to represent this, for this implementation we need to use two bits for each, and add an

extra input Z, also associated with a negative US, where neither A nor B are present. So we have:

Input	CS	US
A	[1 0 0 1]	[1]
B	[0 1 1 0]	[1]
AB	[1 0 1 0]	[0]
Z	[0 1 0 1]	[0]

This scenario was presented to a `Cortex` with an input map of four units, a middle map of two units and the output map, with a single unit.

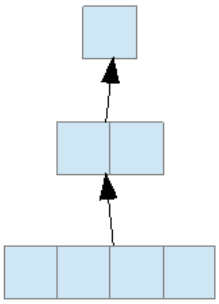


Figure 1: `Cortex` used to run the Simple XOR scenario

The standard or "all equal" version of this scenario presents all four inputs on all the iterations, always on the same order. One variation ("proba") is to give to some of the inputs, typically AB and Z, a probability to appear in any iteration. After trying a few different proportions we settled on a probability of 0.2, i.e., these inputs will appear on $^1/_5$th of the iterations, which was enough to show in a practical time the results we needed. The other variation is randomizing the order of the inputs using a "corpus" (see appendix A.1). Both variations can be combined resulting in a randomized corpus where some of the inputs appear on 20% of the iterations.

The first of these variations was our first test for Learning by heart, with the idea that the inputs that appear only on $^1/_5$th of the iterations will be easier to learn by heart by the `Hippocampus` than for the `Cortex`.

2.4.2 Results

The Graphs The simple XOR, the base or control scenario, was tested with the three brains mentioned on 2.3. These graphs show the output, but before presenting them, we need to clarify what they show.

When we run tests we are limited on what we can "see" of the behaviour of the system, the only output we have is the `Amygdala`'s predictions on each iteration of the simulation. All of our observations of the `Cortex` or the `Hippocampus` are indirect since we don't have an easy way to see what's happening on them. We implemented some probes on the `Hippocampus`, that dump

the state of both `CA3` and `CA1` to text files, but they slow the system too much to use them on long simulations like those needed to finish these scenarios.

On each graph, the X-axis shows the iterations and the Y-axis shows the predictions for each iteration, normally between 0 and 1. There'll be a note whenever any of this is changed. What we want to see is the evolution of the predictions, from the random values on the first few iterations to the convergence to a final prediction.[3] In many cases, there's an intermediate state, while the information received by the `Amygdala` is not enough for a prediction and it *flatlines*, i.e. outputs an useless prediction around 0.5.

Each simulation was run first for a large number of iterations, to make sure we capture all of these states and transitions. Once we have an idea of when these transitions happen, we reduced the number of iterations to a few hundred iterations above that, so all the information is clearly presented and the simulation time is reasonable to run them many times. The graphs presented are for those shorter runs.

So here are the graphs for the Simple XOR, for all three brains. All other experiments will be compared to this one, and these figures will be repeated when needed.

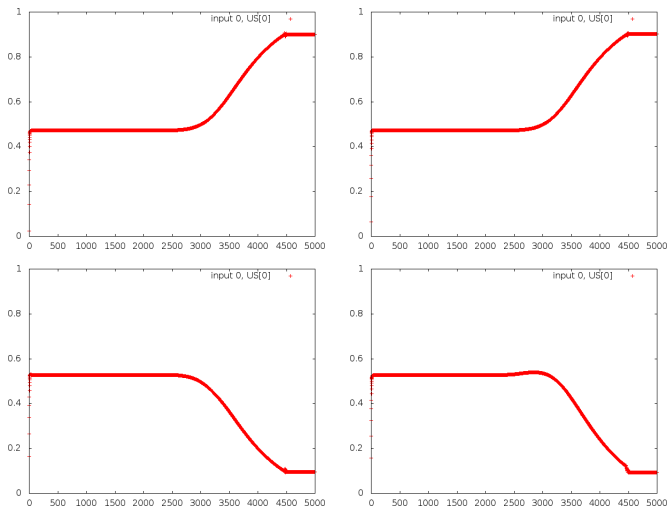


Figure 2: Outputs of Brain CA for the simple XOR showing the *flatline* until around the 2500th iteration and then the transition to the final convergence.
Top row: A and B (US 1), low row: AB and Z (US 0)

[3] There'll be a difference between this prediction and the US determined by the configuration defined threshold for learning

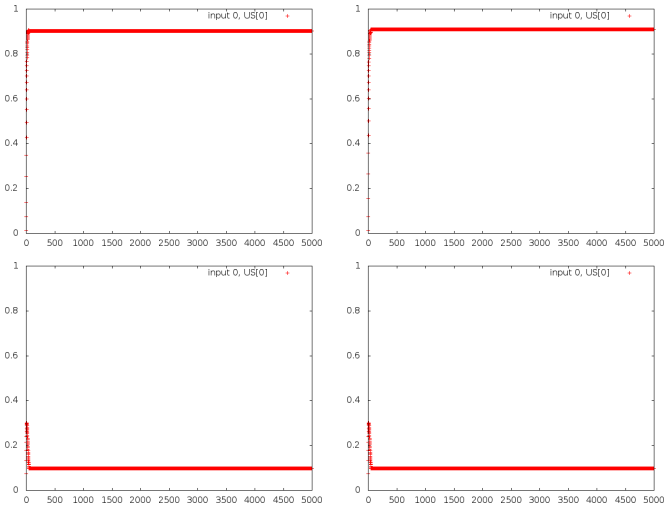


Figure 3: Outputs of Brain HA for the simple XOR showing the almost immediate convergence. Top row: A and B (US 1), low row: AB and Z (US 0)

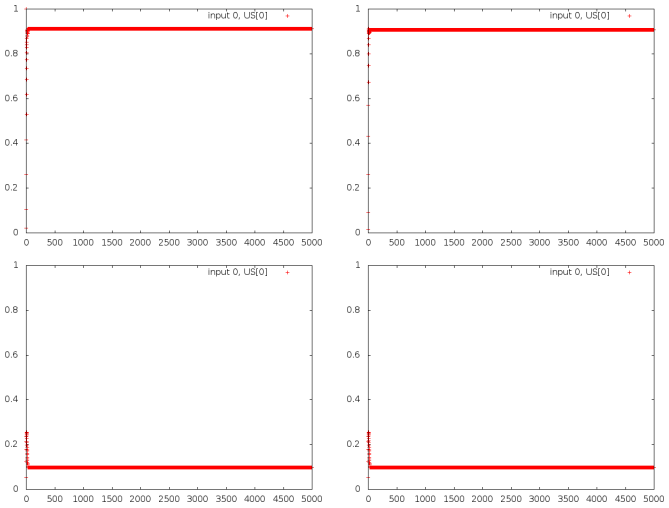


Figure 4: Outputs of Brain CHA for the simple XOR showing the even faster convergence. Top row: A and B (US 1), low row: AB and Z (US 0)

3 Normal working mode

3.1 Introduction

The earlier developments of the `Hippocampus` (as well as the `Cortex`) were focused on some special problems, but the system should be able to work on "normal occasions", and so some properties needed to be tested. We'd like to know what's the *capacity* of the `Hippocampus` and how that capacity scales with the size of the component.

3.1.1 Capacity

The capacity of the `Hippocampus` is how many inputs can be learned without making mistakes on the recall mode. As the number of stimulus stored increases, so does the chance of completing a new one to another already stored. Of course, this limit depends on the *overlap* of the inputs, so when designing a test for this we have to be careful with the randomization and order we use.

3.1.2 Scaling

Once we know the capacity of the `Hippocampus` for inputs of a given size, we would like to know what's the relation between that capacity and that size, whether the former grows linearly with the latter or quadratically or something else.

3.1.3 Learning by heart

One of the functions of the *Hippocampus* is learning by heart some inputs, as opposed to by association and generalization the way the *Cortex* works. We ran some scenarios to see how well the `Hippocampus` implements this behaviour.

3.2 Testing of capacity and scaling

In general, we want to know for an `Hippocampus` of size n, the capacity k of inputs that can be stored before the output is what we call a mistake, according to a given criteria. These inputs will have a number M of bits "on".

For our tests we generate all the $\binom{n}{M}$ possible inputs of size n with M bits on, randomize the order. For these tests we choose $M = n \times 0.15$, based on the work presented on [3] that estimates the normal *Cortex* activity at 15%. Then we start sending them for the `Hippocampus` to learn. After each one is supposedly learned, we test the recall of what's already there. Once the number of stored inputs passes an arbitrary number we start testing the recall of a random sample of them, otherwise it'd take too much time to finish. Since the order of the inputs can change the k obtained, we run this process 50 times, each time with the outputs in different orders, and our results are the statistics of these 50 runs.

Many different criteria for what's a mistake were used:

Ruthless To make comparisions easier, we want to compare binary vectors, and, since the output vector is not binary, first we normalize every element as

$$N_i = \begin{cases} E_i & \text{if } |E_i - R_i| <= \varepsilon \\ \neg E_i & \text{if } |E_i - R_i| > \varepsilon \end{cases}$$

where
R is the received (output) vector
E is the expected (input) vector
N is R normalized
i is the position on the vector
ε is the precision threshold (for these tests we used $\varepsilon = 0.01$)

and then we check if any element of N differs from the corresponding position on the expected result

Number of different bits Here we use the same normalization, count how many bits are different and we call the output a mistake if it has over a given number of different bits from the input (for these tests, $^M/_2$)

Accumulated sum of failures This criteria takes

$$acc = \sum_{i=0}^{size(E)} |E_i - R_i|$$

and calls R a mistake if acc is over a threshold, in this case, 0.4.

To check whether the mistake is made by what's stored on `CA3` or by the transformation learned by `CA1` to return to the `Entorhinal Cortex`, we implemented some probes or dumps of the modules' states. These probes write to a file the inner status of the components, what's stored on CA3, what was the last input and what's the current output. With this tool we can "see" where the output diverged from the input (and expected output) and infer what is the problem. Although none of those dumps are included in this document, for they are large and confusing even if you know exactly what you are trying to see, whenever we state that the mistake was made by `CA1` or `CA3`, we are basing that on an observation of these dumps.

3.2.1 Results

This is the output from the tests for n from 10 to 35, with $M = n \times 0.15$, rounded up, using as an error criteria the accumulated sum of failures with an error threshold of 0.4. For each n and its corresponding M, we show the statistics after 50 runs of the experiment, i.e., after obtaining 50 values of k

```
Step 2: n = 10, M = 2
Statistics for the Hippocampus' capacity (k):
average:       3
min:           3
first_quartil: 3
median:        3
third_quartil: 3
max:           3
All samples, sorted for clarity:
[3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
  3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]

Step 2: n = 15, M = 3
Statistics for the Hippocampus' capacity (k):
average:       3
min:           1
first_quartil: 3
median:        4
third_quartil: 4
max:           4
All samples, sorted for clarity:
[1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4,
  4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]

Step 3: n = 20, M = 3
Statistics for the Hippocampus' capacity (k):
average:       3
min:           1
first_quartil: 4
median:        4
third_quartil: 4
max:           4
All samples, sorted for clarity:
[1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
  4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]

Step 4: n = 25, M = 4
Statistics for the Hippocampus' capacity (k):
average:       3
min:           1
first_quartil: 2
median:        4
third_quartil: 4
max:           4
All samples, sorted for clarity:
[1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4,
  4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]

Step 5: n = 30, M = 5
```

```
Statistics for the Hippocampus' capacity (k):
average:       2
min:           1
first_quartil: 2
median:        3
third_quartil: 4
max:           4
All samples, sorted for clarity:
[1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3,
  3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]

Step 6: n = 35, M = 6
Statistics for the Hippocampus' capacity (k):
average:       2
min:           1
first_quartil: 1
median:        2
third_quartil: 3
max:           5
All samples, sorted for clarity:
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5]

Step 7: n = 40, M = 6
Statistics for the Hippocampus' capacity (k):
average:       2
min:           1
first_quartil: 1
median:        3
third_quartil: 3
max:           5
All samples, sorted for clarity:
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3,
  3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5]
```

3.2.2 Discussion

Our experience with the model shows that it and, more precisely, the `Hippocampus` + `Amygdala` can handle more inputs that what these tests show. The scenarios that we used for the rest of our tests, as we'll show below, store 4, 8 and 16 different stimuli with $n = 4$ on the first case and $n = 8$ on the other two, while these tests show from 3 inputs for $n = 10$ to at most 5 elements for $n = 40$. The problem with the tests is that the criteria used to decide what's a mistake are artificial and too strict compared to the complete system. Since the latter is dynamic, it adapts to those mistakes or differences with the theoretical binary values which makes the capacity higher.

Using probes, i.e. dumping the states to files as mentioned on 3.2, we could see that when the tests report an error, the output from `CA3` would be considered valid for any of the criteria. The differences appear when `CA1` tries to reconvert `CA3`'s output to the original input. Apparently

learning and unlearning the conversions fast and precisely enough it's hard for `CA1`.

We didn't test with bigger values of n because that starts to excede the computer's capacity, since the matrix on `CA3` store $O(n^4)$ elements (the `DG` translates from n to $\frac{n_i^2-n_i}{2} \in O(n^2)$ and that is the width and height of the matrix), so testing with higher values of n becomes too slow and impractical. Since the matrix stores mostly 0s, a better implementation can help with this problem, but the main issue of the definition of "mistake" remains.

Also the numbers themselves start to be intractable. For example, calculating the number of inputs that will be generated (which we do to reserve all memory at once, i.e. faster than one portion at a time) for which we need to calculate $n!$, can't be done for $n \geq 34$ because 34! is larger than 64 bits.

3.3 Testing for Learning by heart

This scenario consists of two simple XORs adjacent to one another, with the idea of presenting one of them less than the other and expecting that the less frequent is learned by heart but not by the `Cortex`. The inputs for this scenario are:

Input	CS	US
A1	`[1 0 0 1 0 0 0 0]`	`[1]`
B1	`[0 1 1 0 0 0 0 0]`	`[1]`
AB1	`[1 0 1 0 0 0 0 0]`	`[0]`
Z1	`[0 1 0 1 0 0 0 0]`	`[0]`
A2	`[0 0 0 0 1 0 0 1]`	`[1]`
B2	`[0 0 0 0 0 1 1 0]`	`[1]`
AB2	`[0 0 0 0 1 0 1 0]`	`[0]`
Z2	`[0 0 0 0 0 1 0 1]`	`[0]`

The `Cortex` for this has eight units on the input map, two middle maps of two units each and finally the out level with a single unit. We tried this scenario first with all inputs on each iteration, to see how the system behaves with it. Then we ran simulations giving the second half (A2, B2, AB2 and Z2) a probability of 0.2 to appear and finally we run this again, but this time randomizing the order of the stimuli.

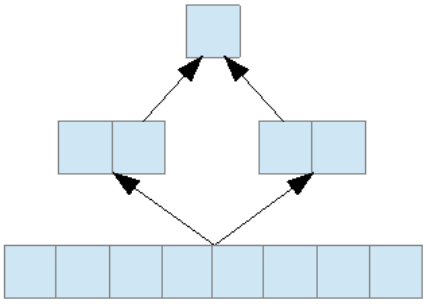


Figure 5: `Cortex` used to run the Double XOR scenario

3.3.1 Results

Simple XOR, using probabilities This is the same scenario, but while the inputs A and B are presented on every iteration, AB and Z have only a probability of 0.2 of being presented on each one. The scale for the X axis on those two graphs is in a way different and in another the same. It shows the number of the iterations where each stimulus was presented, so where the last datapoint for AB or Z is 200 this means that that stimulus appeared 200 times, not that it appeared last on the 200th iteration. In this sense of the absolute iteration numbers is different, but considering the real time of the simulation, the values to the right of the graphs where computed roughly at the same time for each one. There are a few additional artifices produced by the autoscaling from `gnuplot`, the program used to generate these graphs.

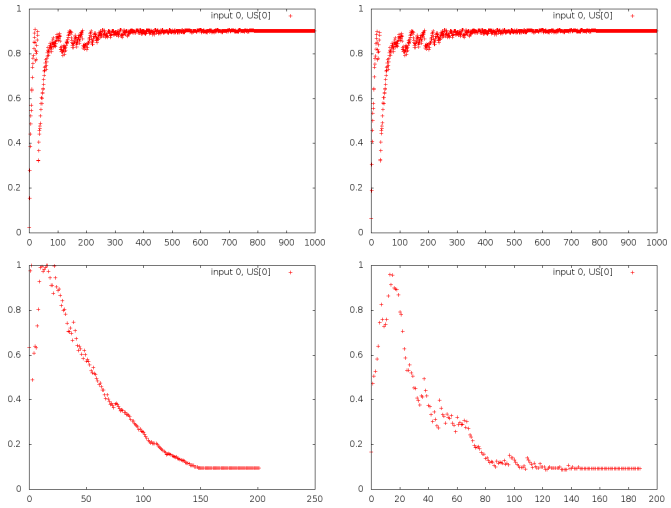


Figure 6: Outputs of Brain CA for the simple XOR where AB and Z have 20% chance of appearing on each iteration, showing a faster convergence, without *flatline* for all inputs. Top row: A and B (US 1), low row: AB and Z (US 0)

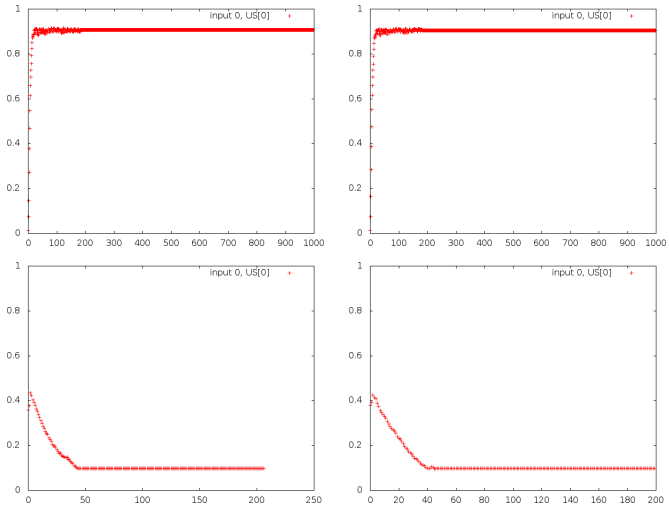


Figure 7: Outputs of Brain HA for the simple XOR where AB and Z have 20% chance of appearing on each iteration, showing a little interference on the first iterations of A and B, and, thanks to the lower number of iterations, more detail on the convergence for the other two inputs.
Top row: A and B (US 1), low row: AB and Z (US 0)

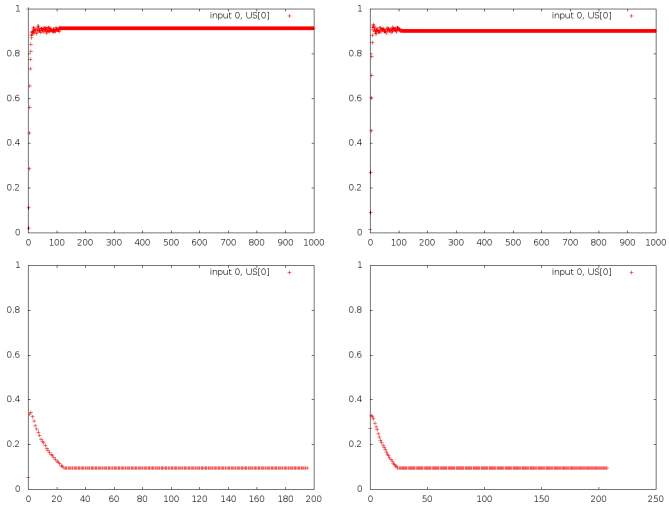


Figure 8: Outputs of Brain CHA for the simple XOR where AB and Z have 20% chance of appearing on each iteration, showing again that the convergence is faster.
Top row: A and B (US 1), low row: AB and Z (US 0)

Simple XOR, randomized Again the same simple scenario, but this time we feed the stimuli in random orders, first with equal probabilities for all of them:

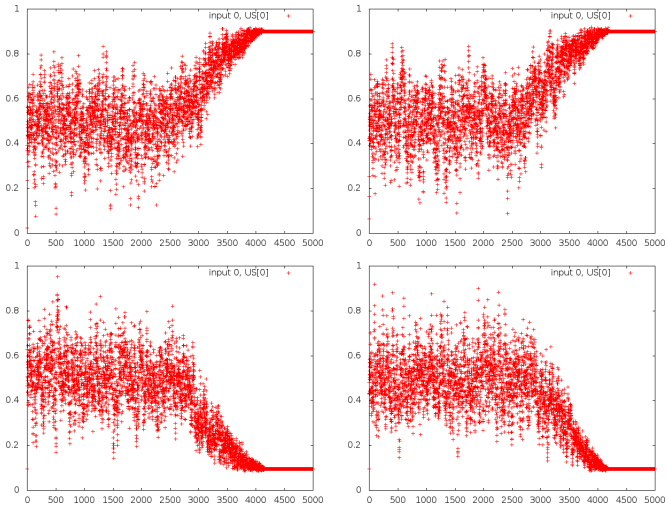


Figure 9: Outputs of Brain CA for the simple XOR, with the inputs randomized. The randomization changes the *flatline* state by random noise, until the `Cortex`'s activity is strong enough to generate a convergence.
Top row: A and B (US 1), low row: AB and Z (US 0)

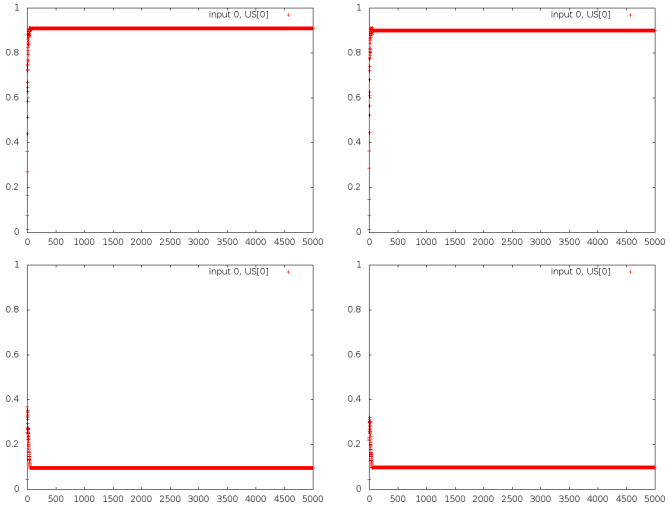


Figure 10: Outputs of Brain HA for the simple XOR, with the inputs randomized. This graph is practically identical to 3, the randomization doesn't affect the `Hippocampus`
Top row: A and B (US 1), low row: AB and Z (US 0)

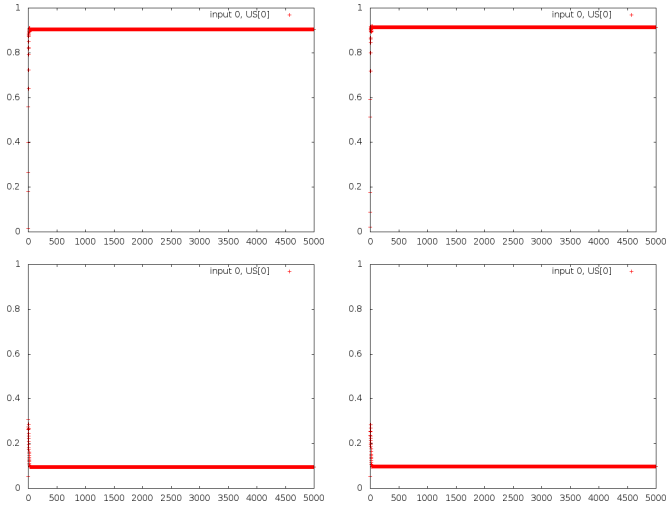


Figure 11: Outputs of Brain CHA for the simple XOR, with the inputs randomized. This graph is also identical to 4.
Top row: A and B (US 1), low row: AB and Z (US 0)

and then with the same disproportion (5:1) as before

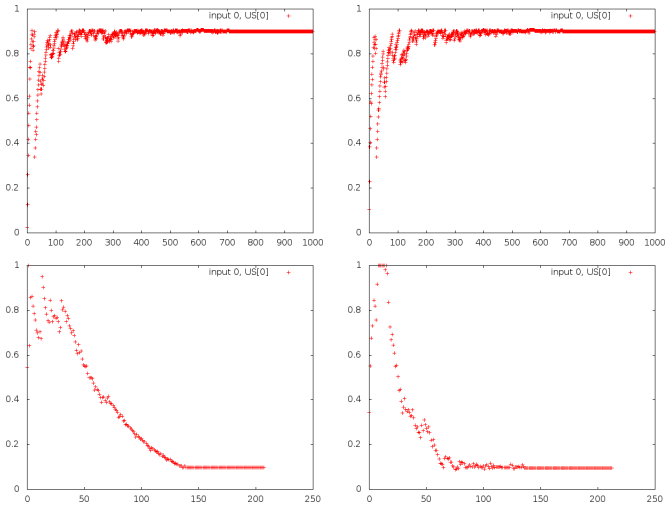


Figure 12: Outputs of Brain CA for the simple XOR, with the inputs disproportionate and randomized. This removes the noise shown in the figure 9, but the evolution to the convergence is noiser than in 6 for all four inputs.
Top row: A and B (US 1), low row: AB and Z (US 0)

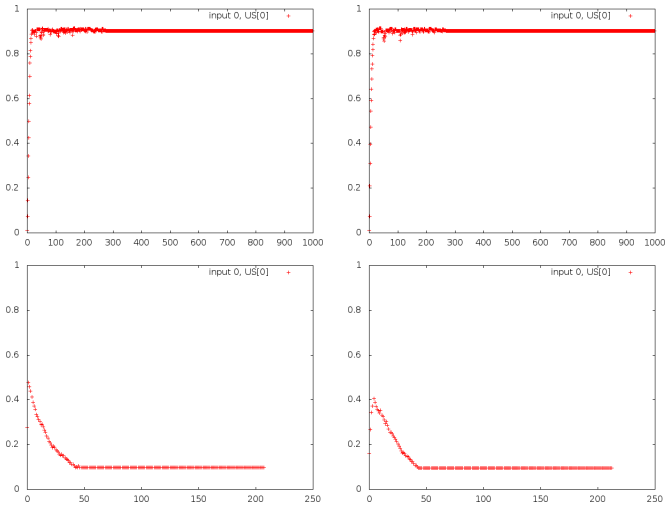


Figure 13: Outputs of Brain HA for the simple XOR, with the inputs disproportionate and randomized. It's basically a noisier version of fig. 7
Top row: A and B (US 1), low row: AB and Z (US 0)

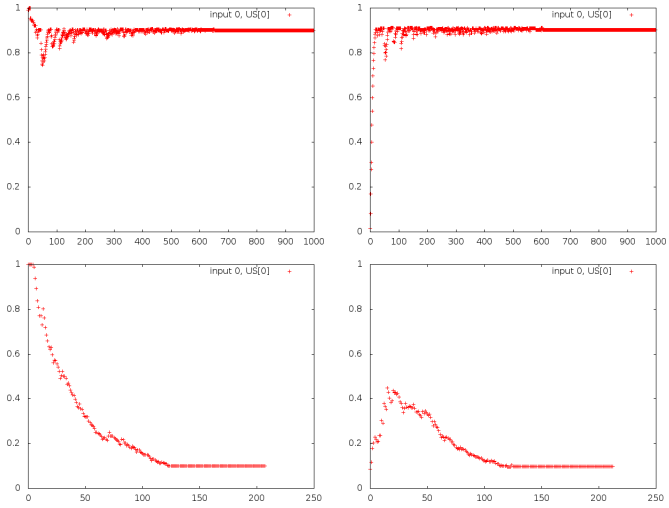


Figure 14: Outputs of Brain CHA for the simple XOR, with the inputs disproportionate and randomized. It's noisier than the figures 8 and 13, the cortex's influence is "negative" here. Top row: A and B (US 1), low row: AB and Z (US 0)

Double XOR As mentioned on the previous section, we first ran this scenario with all stimuli in every iteration

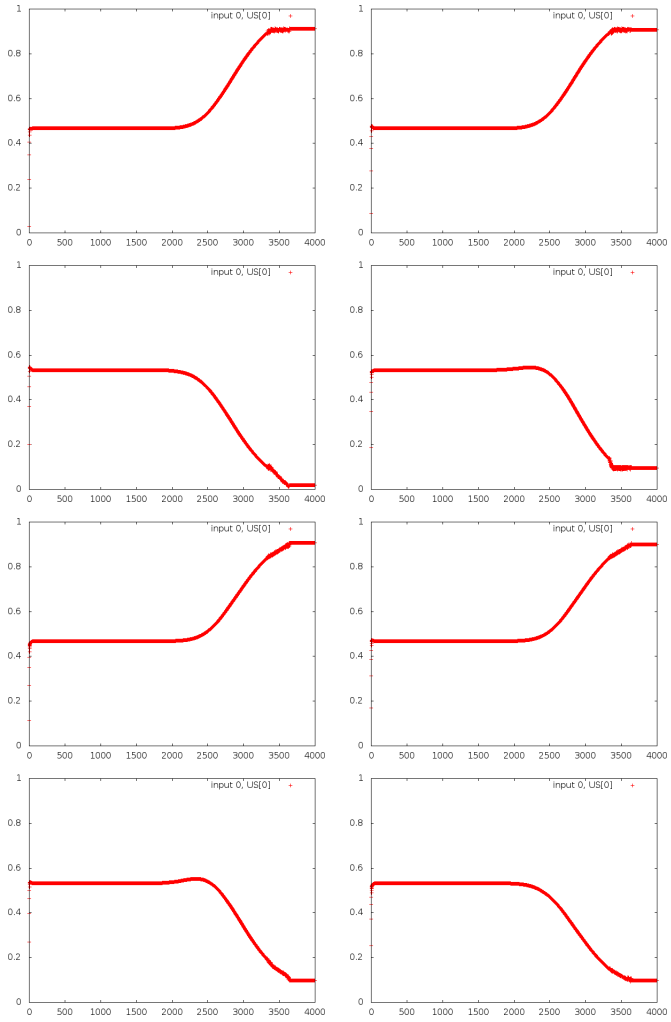


Figure 15: Outputs of Brain CA for the double XOR, showing that both "halves" behave independently as two simple XORs for this brain, since the first four graphs are equal to figure 2 as are the second four.
First row: A_1 and B_1 (US 1), second row: AB_1 and Z_1 (US 0), third row: A_2 and B_2 (US 1), last row: AB_2 and Z_2 (US 0)

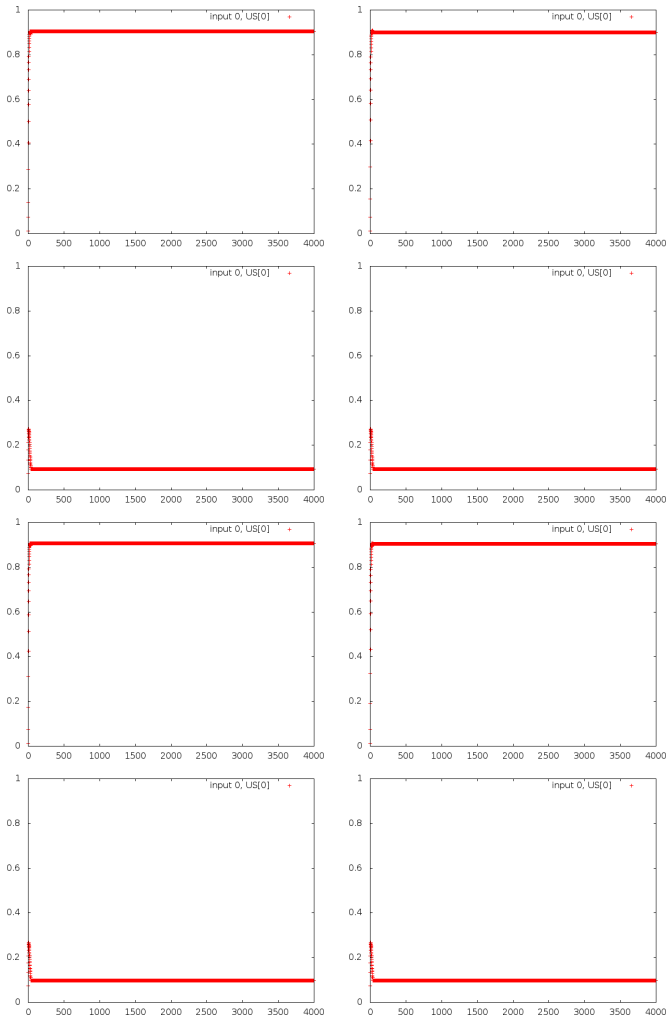


Figure 16: Outputs of Brain HA for the double XOR, also showing that both "halves" behave independently as two simple XORs for this brain.
First row: A_1 and B_1 (US 1), second row: AB_1 and Z_1 (US 0), third row: A_2 and B_2 (US 1), last row: AB_2 and Z_2 (US 0)

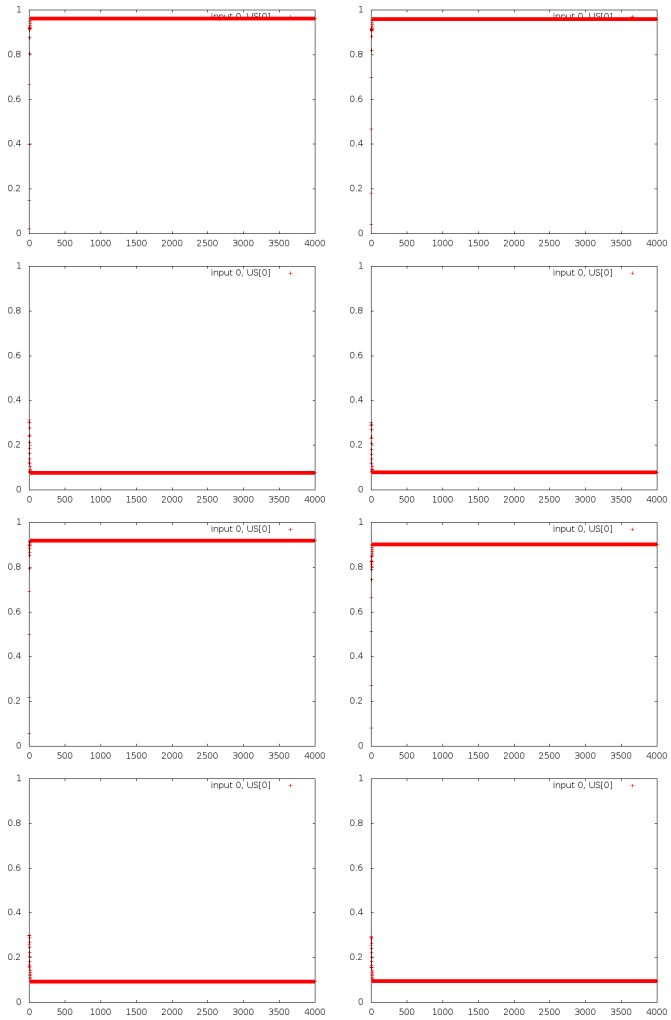


Figure 17: Outputs of Brain CHA for the double XOR, also showing that both "halves" behave independently as two simple XORs for this brain.
First row: A_1 and B_1 (US 1), second row: AB_1 and Z_1 (US 0), third row: A_2 and B_2 (US 1), last row: AB_2 and Z_2 (US 0)

And then with the second half appearing around $^1/_5$th of the iterations.

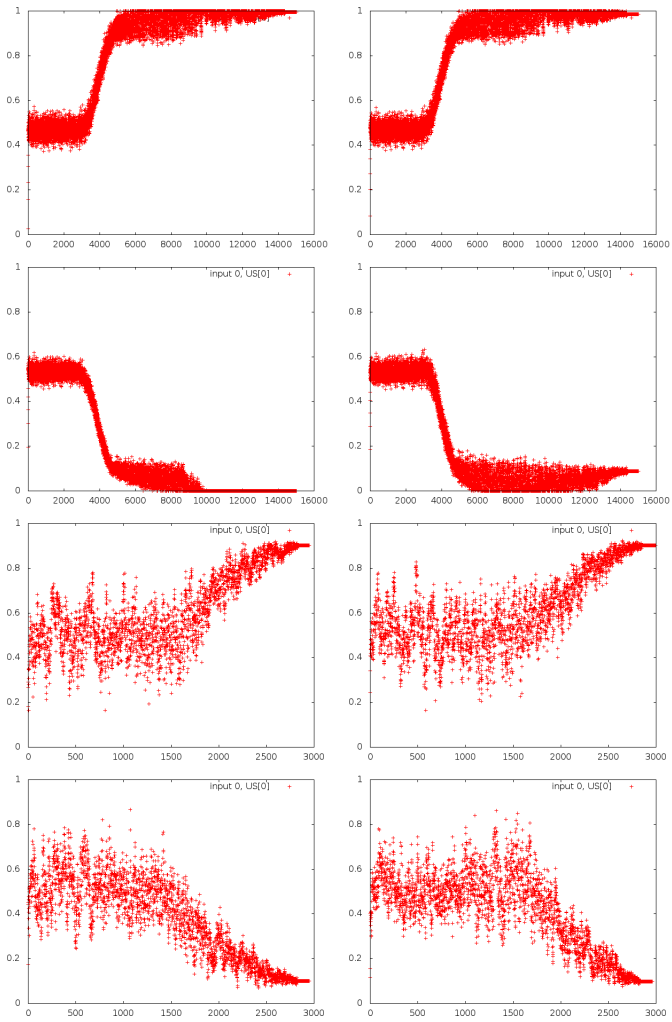


Figure 18: Outputs of Brain CA for the double XOR, where A_2, B_2, AB_2 and Z_2 have 20% chance of appearing on each iteration. This generates a different kind of noise, as if the inputs were randomized.
First row: A_1 and B_1 (US 1), second row: AB_1 and Z_1 (US 0), third row: A_2 and B_2 (US 1), last row: AB_2 and Z_2 (US 0)

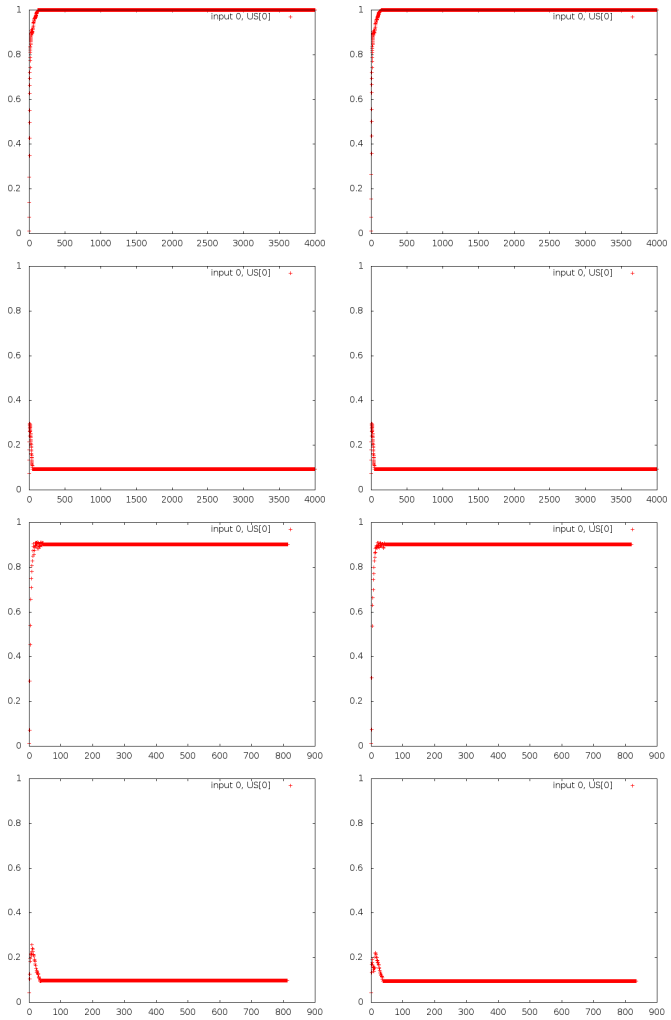


Figure 19: Outputs of Brain HA for the double XOR, where A_2, B_2, AB_2 and Z_2 have 20% chance of appearing on each iteration. These show little difference with the previous results (fig. 16).
First row: A_1 and B_1 (US 1), second row: AB_1 and Z_1 (US 0), third row: A_2 and B_2 (US 1), last row: AB_2 and Z_2 (US 0)

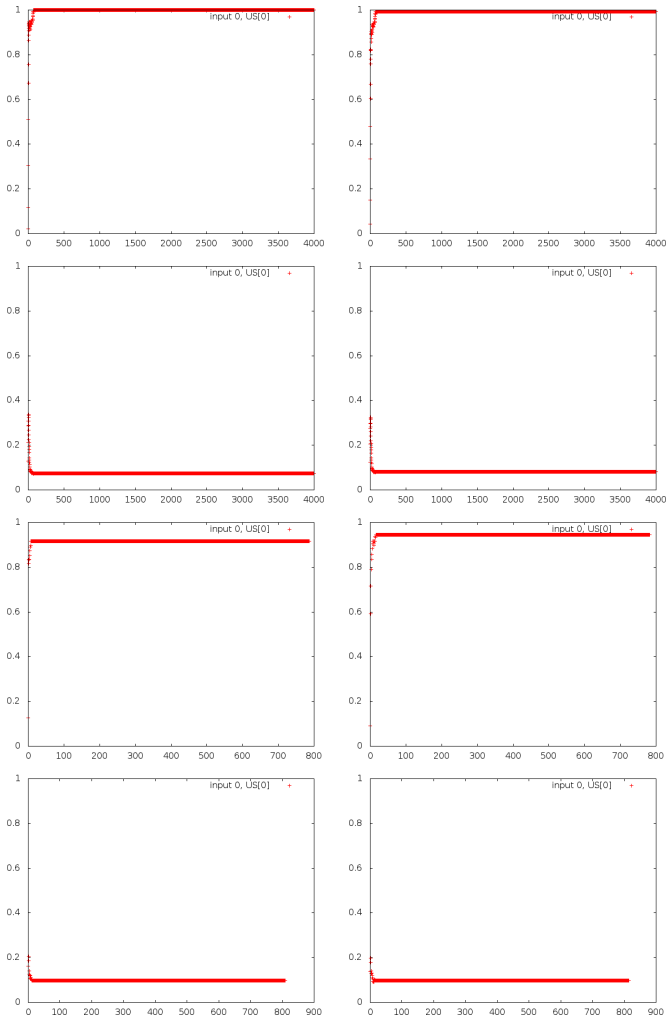


Figure 20: Outputs of Brain CHA for the double XOR, where A_2, B_2, AB_2 and Z_2 have 20% chance of appearing on each iteration. Again, no difference with the previous results). First row: A_1 and B_1 (US 1), second row: AB_1 and Z_1 (US 0), third row: A_2 and B_2 (US 1), last row: AB_2 and Z_2 (US 0)

Double XOR with randomness Running the same scenario but randomizing the inputs can generate very different outputs in different simulations using the Brain without `Hippocampus`, some of which we present here. This has no relation with learning by heart, the results from this scenario on the brains HA and CHA don't show anything new and we won't even present them here. However, the effect of this randomness on the CA might be interesting to someone working on the `Cortex`, so we present that as data, without too much analisys.

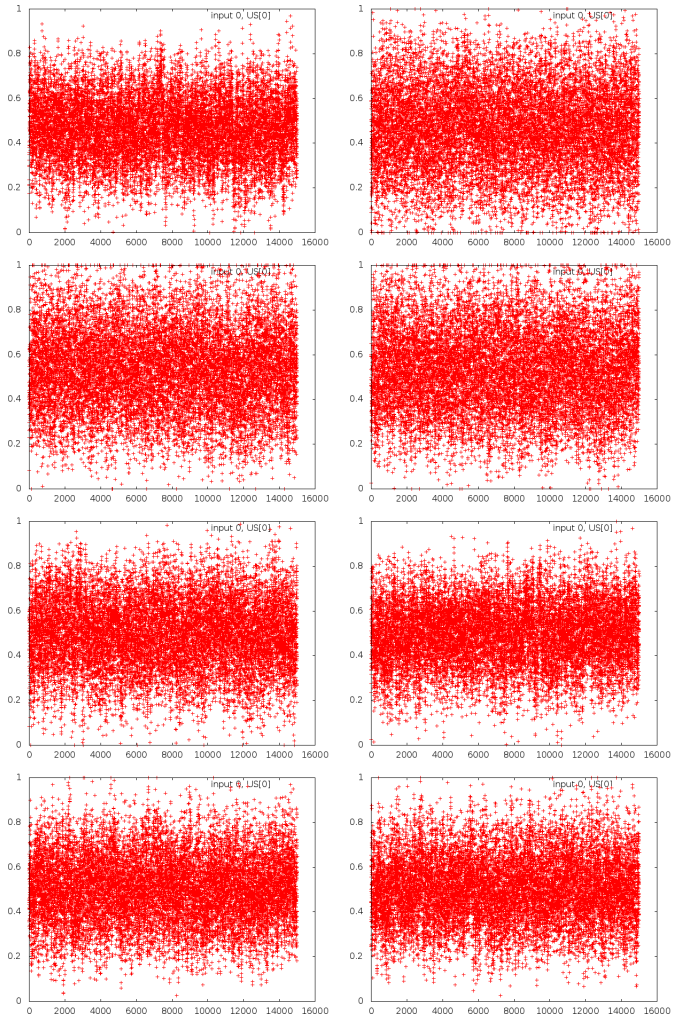


Figure 21: Outputs of Brain CA for the double XOR randomized, in a simulation that didn't converge. Notice the large number of iterations.
First row: A_1 and B_1 (US 1), second row: AB_1 and Z_1 (US 0), third row: A_2 and B_2 (US 1), last row: AB_2 and Z_2 (US 0)

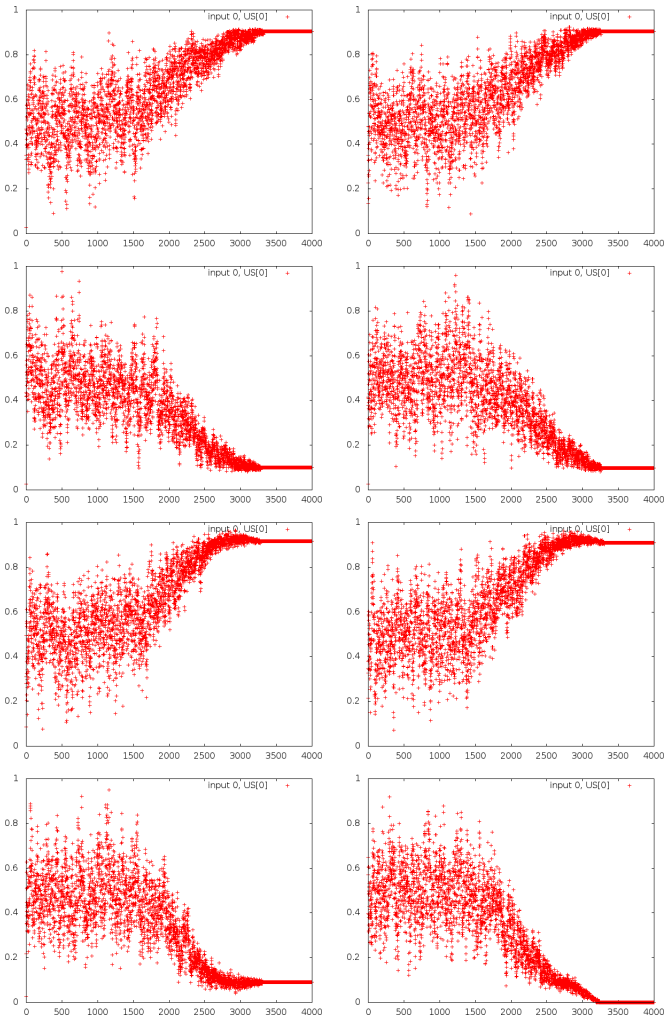


Figure 22: Outputs of Brain CA for the double XOR randomized, in a simulation that converged in about 3500 iterations. This is similar to 18
First row: A_1 and B_1 (US 1), second row: AB_1 and Z_1 (US 0), third row: A_2 and B_2 (US 1), last row: AB_2 and Z_2 (US 0)

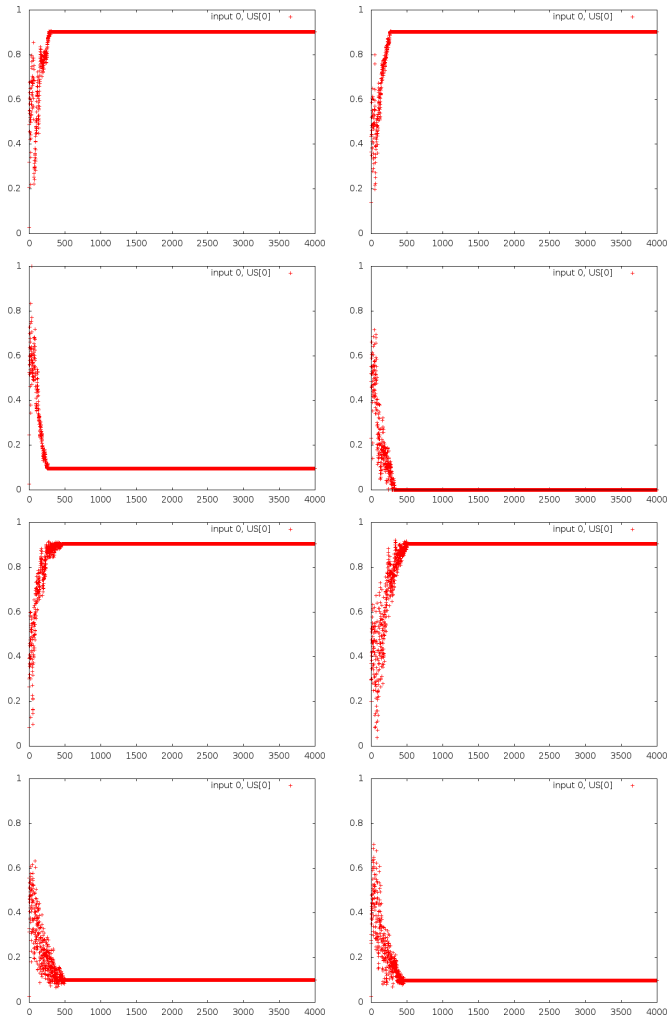


Figure 23: Outputs of Brain CA for the double XOR randomized, in a simulation that converged very fast.
First row: A_1 and B_1 (US 1), second row: AB_1 and Z_1 (US 0), third row: A_2 and B_2 (US 1), last row: AB_2 and Z_2 (US 0)

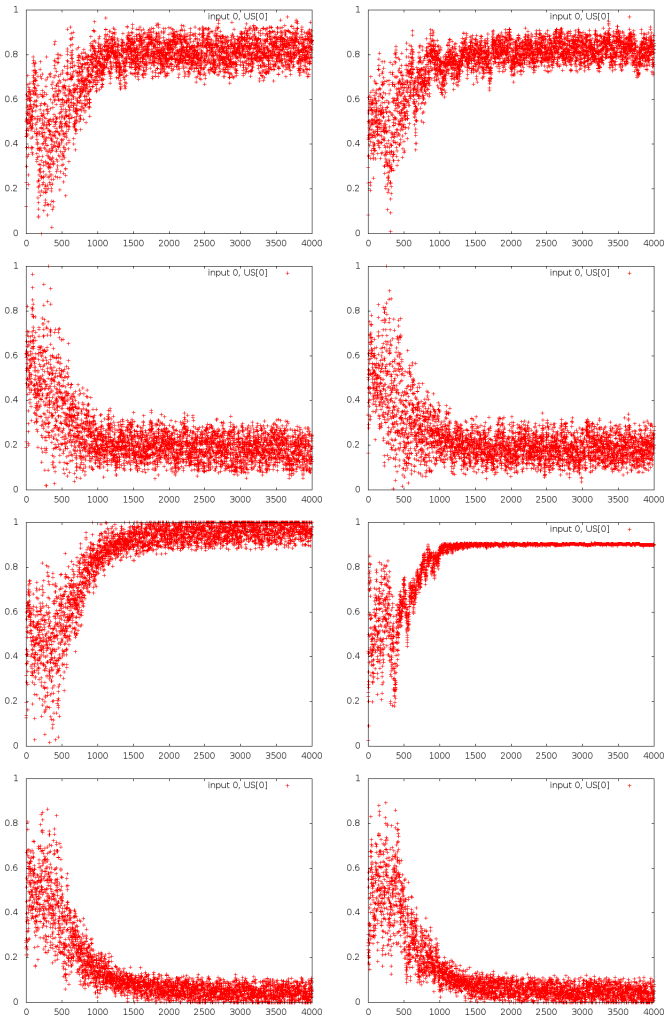


Figure 24: Outputs of Brain CA outputs for the double XOR randomized. These graphs aren't similar to any other, except for the one corresponding to B_2.
First row: A_1 and B_1 (US 1), second row: AB_1 and Z_1 (US 0), third row: A_2 and B_2 (US 1), last row: AB_2 and Z_2 (US 0)

3.3.2 Discussion

The results of these tests don't really say much about the behaviour of the `Hippocampus`. It does what it's supposed to do, learns things fast and helps the `Amygdala` to converge to a solution in only a few iterations, two orders of magnitude faster than the `Cortex` alone. But this quickness doesn't help testing how it behaves with regards to learning by heart. All of the graphs for the brains with `Hippocampus` have the same general form: 40 to 70 iterations of convergence and then the predictions stay at the same value for the rest of the simulation.

That being said, these tests show some interesting behaviours from the brain CA, the one with only a `Cortex` and a `Amygdala`, on cases that the previous work didn't cover cases like these, so we'll discuss them here.

`Cortex` behaviour on disproportionate scenarios As shown on the figure 6, when AB and Z appear $^1/_5$th of the time the `Cortex` is able to converge to a solution much faster, near to 5 times faster. We believe that the disproportion allows the component to generate a representation for A and B faster, since AB and Z interfere less with the process. To compare, we inverted the disproportion, presenting AB and Z on all iterations and A and B with probability 0.2. The brains with an `Hippocampus` can solve this without problems, but for brain CA we got these results which again we present mostly as data to be analized somewhere else:

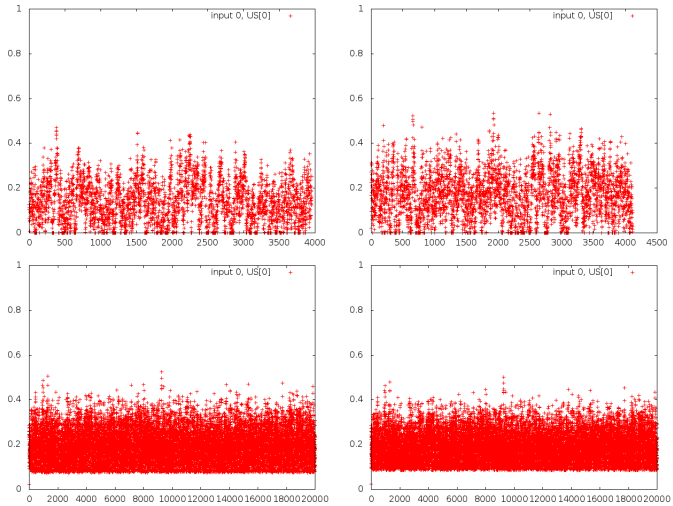


Figure 25: Outputs of Brain CA for the simple XOR, with the A and B appearing $^1/_5$th of the iterations instead of AB and Z. After 20000 iterations, no convergence was reached. Top row: A and B (US 1), low row: AB and Z (US 0)

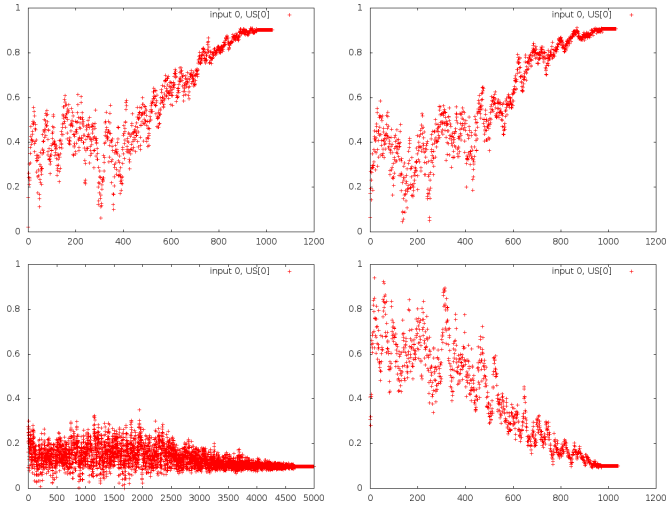


Figure 26: Outputs of Brain CA for the simple XOR, with only AB having probability 0.2 of appearing. This reaches a convergence, after much noise
Top row: A and B (US 1), low row: AB and Z (US 0)

Simple XOR, randomized On the figure 9 is clear that the order on which the stimuli appear matter for the `Cortex`. Here, where A, B, AB and Z are presented randomly (but the same amount of times each one) there's just noise until the convergence. The presence of an `Hippocampus` on the brain smoothes that randomness and the predictions are again fast.

Using both the randomization and the disproportion we remove most of the noise and the results presented on the figure 12 are similar to 6. This again implies that the least the contradictory stimuli appear (AB and Z contradict what can be learned from the presence of A and B), the least it interferes with the `Cortex`'s (and/or the `Amygdala`'s) convergence. This idea is further supported by the next two graphs, where the disproportion is applied only to AB and not Z:

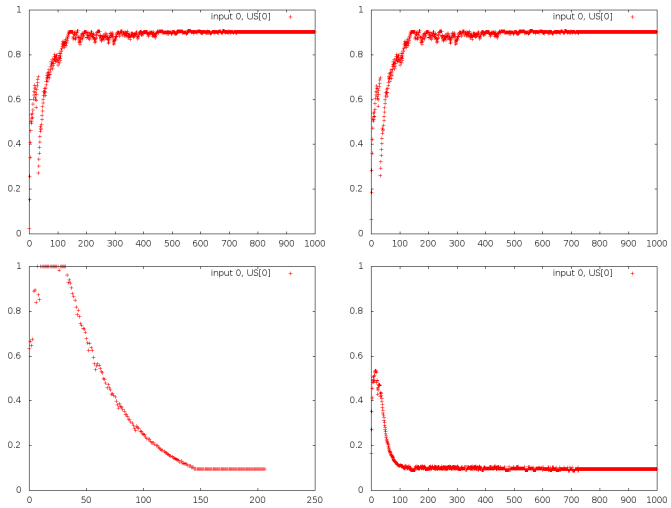


Figure 27: Brain without Hippocampus' outputs for the simple XOR with different probabilities on AB only Top row: A and B (US 1), low row: AB and Z (US 0)

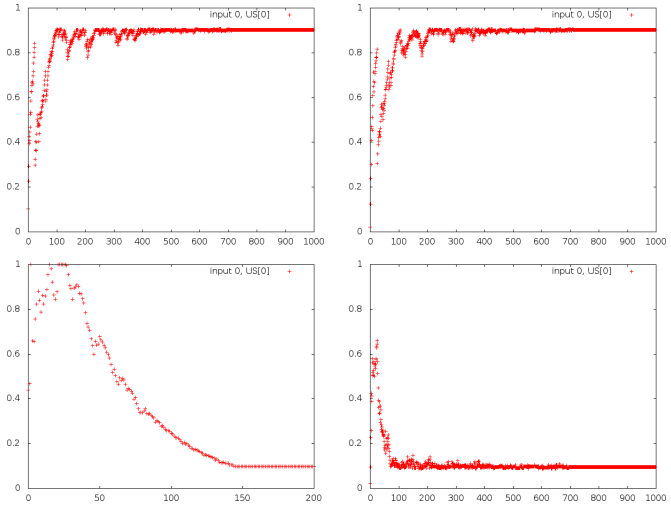


Figure 28: Brain without Hippocampus' outputs for the simple XOR with different probabilities on AB only and presented randomly
Top row: A and B (US 1), low row: AB and Z (US 0)

Double XOR Presenting this scenario straight to the different brains produces outputs no different from presenting simple XORs we've seen before. However, once we start modifying the simulation's parameters, the output from the brain with `Cortex` only changes substantially. If we present the second half fewer times the brain takes from two to three times as many iterations to converge[4]; the predictions for the first half (the one that always appears) are noisy and don't even converge at the same time; and for the second half there's lots of noise until it starts slowly converging.[5]

When we add randomness to the "mix", the variations on the behaviour increase. Using only randomness (i.e. all stimuli presented on all iterations) It can either generate pure noise even after 15000 iterations (fig. 21), converge in a "reasonable" number of iterations (fig. 22), converge quickly and almost without noise (fig. 23 and even generate results that are just plain strange (fig. 24). This variety of results is a constant, we didn't have to run lots of simulations to find them. Most simulations generate results with some noise and then a convergence, in any given number of iterations, but about one in ten generate strange results or just noise. Using randomness with not equiprobable stimuli generated no new results, so we don't present those graphs here.

Again we don't have any theory, nor even an intuition of why the `Cortex` + `Amygdala` give any of these results, since most of our work was on the `Hippocampus`. Speaking of which, the brains with Hippocampus, again, ignore this randomness and converge to predicting correct results in only a few hundred iterations.

[4] The graph on the figure 18 shows a run that took about 15000 iterations, but due to the randomness of the starting weights on the `Cortex` other runs have taken under 8000

[5] This is second half is aesthetically the most beautiful graph of them all.

4 Forgetting on the Hippocampus

The `Hippocampus` acts as a storage space that keeps stimuli (CS) and reactions (US) as they come, without building associations the way the `Cortex` does. It sounds reasonable that, given that its capacity is limited, it should have some mechanic to *forget*, to empty the data stored that it has stored to make room for new inputs. It's not clear that the *Hippocampus* does indeed forget, and if it does, is definitely not clear how.

We decided to implement and test some mechanics to model this new mode of work on the `Hippocampus`, without trying to decide when they should run, how strongly, etc. This section will introduce these mechanics in the order they were created, starting with the simplest and growing in complexity, until we reach the "final" model.

4.1 Detailed model of `CA3`

The data storage of the `Hippocampus` is `CA3`, so this is what we needed to modify to implement forgetting. To better explain the new model let's detail how this component works in the modes already implemented:

Learning The input to `CA3` is a vector $input \in \{0,1\}^n$, for a given size n. The system is implemented with real numbers, and the outputs are usually real numbers. However, accepting real numbers for inputs increases the difficulty of deciding if an input is new or just a version of what we already new. We'll discuss more about this issue latter, but for now we need to clarify that some of the equations that govern this module assume that the inputs are either 0 or 1.

When along with the input comes the signal to store that input, this is stored in a matrix $W \in \mathbb{R}^{n \times n}$:

$$\Delta W_{ij} = input_i \times input_j$$

Therefore, inputs are redundantly stored, so for example:

$$i = \begin{bmatrix} 0 & 1 & 0 & 1 \end{bmatrix}$$

is stored as

$$W = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

Recall The *output* vector is a function of both the *input* and the contents of the matrix W, created from the memories stored. This process happens "always", i.e.: all inputs create an output, regardless of whether the input is learned or not.

Before the implementation of forgetting the equation for the output was:

$$\Delta output_j = \frac{-output_j + h(\sum_i W_{ij} * input_i - \sum_i input_i)}{\tau} \quad (1)$$

where

$$h(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Continuing with the previous example, and ignoring the dynamic nature of the computatin, we get for the same $i = [0\ 1\ 0\ 1]$.

$$\begin{aligned} output_1 &= -0 + h(0 - 0) = 0 \\ output_2 &= -0 + h(2 - 2) = 1 \\ output_3 &= -0 + h(0 - 0) = 0 \\ output_4 &= -0 + h(2 - 2) = 1 \end{aligned}$$

and if we used $i = [0\ 1\ 0\ 0]$ we'd get

$$\begin{aligned} output_1 &= -0 + h(0 - 0) = 0 \\ output_2 &= -0 + h(2 - 2) = 1 \\ output_3 &= -0 + h(0 - 0) = 0 \\ output_4 &= -0 + h(1 - 1) = 1 \end{aligned}$$

completing the original input stored.

This output could be compared with the input and use that difference as an indicator of the novelty of an input for the separation/completion problem, with a higher difference showing that the stimulus is likely to be new.

4.2 Mechanics of forgetting

4.2.1 Global (systematic) forgetting

With the first and simplest mechanic implemented, all the memories stored are gradually forgotten over time. On `CA3`, for each unit on the matrix W_{ij} that stores data (i.e. has a non-zero value) we'll have

$$\Delta W_{ij} = \frac{f(W_{ij})}{\tau} < 0$$

The simplest f is applying a constant forgetting rate φ:

$$\Delta W_{ij} = \frac{-W_{ij} \times \varphi}{\tau}$$

(notice that even though in the previous examples we were using binary values, the implementation of W uses `doubles`)

4.2.2 Dynamic forgetting

However, discussing what would be needed to implement other features missing from the model (notably consolidation, described on 6.2), it became clear that a global mechanic might not be enough. We'll need to be able to forget a particular example, whether as the only way of forgetting or alongside the global leak. For this model we decided to go with the latter option, and use the global forgetting as a general mechanic, modulated by **selective forgetting**.

At the same time we realized that memories aren't all equal and so it would be interesting to model the **strength of the memories**. In our current model all inputs are stored on the `Hippocampus` and with the same intensity. But when the error in the prediction for a stimulus is large (for whatever value of large), or perhaps larger than the errors produced for the rest of the problem, it is more important that the `Hippocampus` learns them by heart. The memory should be stronger. This strength can be modeled as another modulation of the general forgetting: stronger stimuli will have lower forgetting rate.

So now instead of a global rate φ we need to have a different rate for each input, stored in a new matrix F where, for each W_{ij} that has a value (i.e. is part of a remembered input), we have an associated forgetting rate φ_{ij} that will go up with selective forgetting, and down with the reappearance of the stimuli. In this latter case, the reduction of the corresponding φ_{ij} will be dependent on the required strength of the memory while for the former we used a constant value for the increment, which will make φ_{ij} fall exponentially over time. If we plot W_{ij} over time, φ_{ij} would be the absolute value of the slope of that plot. Since we'll never want the W_{ij} to grow on its own, that slope is always < 0.

The mechanic for forgetting in any cell is

$$\Delta W_{ij} = \frac{-W_{ij} \times \varphi_{ij}}{\tau} < 0$$

Recalling an input inhibits forgetting it, and once an output starts to converge due to completion, forgetting is also inhibited for those cells. Each φ_{ij} will go up and down as just described with the following operations:

- **Forget all** Reduces all memories according to their corresponding rates.

$$\Delta W_{ij} = \frac{-W_{ij} \times \varphi_{ij}}{\tau}$$

- **Forget all except *input*** runs the same operation, but the reduction of the W_{ij} corresponding to the input is inhibited. More precisely, the previous item is not applied if $input_i \times input_j \neq 0$

- **Forget *input*** increases the forgetting rate for the input and applies the mechanic for it. I.e., if $input_i \times input_j \neq 0$:

$$\Delta \varphi_{ij} = \frac{\varphi_0}{\tau}$$

$$\Delta W_{ij} = \frac{-W_{ij} \times \varphi_{ij}}{\tau}$$

  where φ_0 is the base forgetting rate, which is a configurable constant.

- **Learn *input*, *s*** increases the value of the memory (the corresponding W_{ij}) by the given strength s and modifies the φ_{ij} accordingly.

  $$\Delta W_{ij} = \frac{W_{ij} \times s}{\tau}$$

  The first time an input is presented (i.e. $W_{ij} = 0$), a value of 1 is used instead of $W_{ij} \times s$. At the same time, φ_{ij} is set or initialized:

  $$\varphi_{ij} = \begin{cases} \frac{\varphi_0}{s} & \text{if } \varphi_{ij} = 0 \\ min(\varphi_{ij}, \frac{\varphi_{ij}}{s}) & \text{if } \varphi_{ij} > 0 \end{cases}$$

  Again, this is only applied where $input_i \times input_j \neq 0$

4.3 Modifications to the recall mode

The equations for recall assume that the contents of W are integer numbers, usually 0 or 1. As soon as you add a leak, that's no longer true, and so, since the inputs are still binaries, $\sum_i W_{ij} * input_i < \sum_i input_i$, $h()$ always returns 0 and there's no output. To avoid this, we changed the recall equation 1 to

$$\Delta output_j = \frac{-output_j + \sum_i W_{ij} * input_i}{\tau}$$

But now the outputs stop being around 1 and 0, so to "normalize" the output we add back the $\sum_i input_i$ term.

$$\Delta output_j = \frac{-output_j + \frac{\sum_i W_{ij} * input_i}{\sum_i input_i}}{\tau}$$

Now the output is no longer a vector in $\{0,1\}^n$ but an approximation, so someone should make a decision about when a number is "1" or not. Right now that component is `CA1` without modifications. `CA1` learns to translate from the "language" used in `CA3` to the one used on the rest of the brain, and does that by comparing `CA3`'s output with the input to the `Hippocampus`. Since this process is learned, it doesn't actually matter if the output is binary or real. In fact the input to the `Hippocampus` is not necessarily binary either. `CA1` has a large learning rate, so it adjust to the variations quicly, and the output reflects the memory losses with lower activities. We also tried implementing a leak on its neurons, but it didn't add to the model. This component also has some potential problems that we'll dicuss in the section 6.6

The testing protocols for this portion of the research are described on the section 4.4, the results are on 4.4 and discussed on 4.5

4.4 Testing and results

This is more a proof of concept than a complete test of the behaviour of the system for a given scenario. We just want to show how our mechanics work compared to the earlier model, so we

ran the simple XOR with an instance of the system with forgetting enabled, and with various forgetting rates (φ_0) looking for a behaviour we'd like for this feature. Although all of the system is implemented, in our tests only the global forgetting is used, since there's still no point where selective forgetting nor strengthed learning are used, so this configured rate is used only as the global leak rate.

Before showing the results with different values of φ_0, we present again the outputs from the three brains for the Simple XOR (figures 3 and 4), as a reference, since thats the "original" behaviour we are comparing to. The output from Brain CA are omitted since these modifications don't afect it.

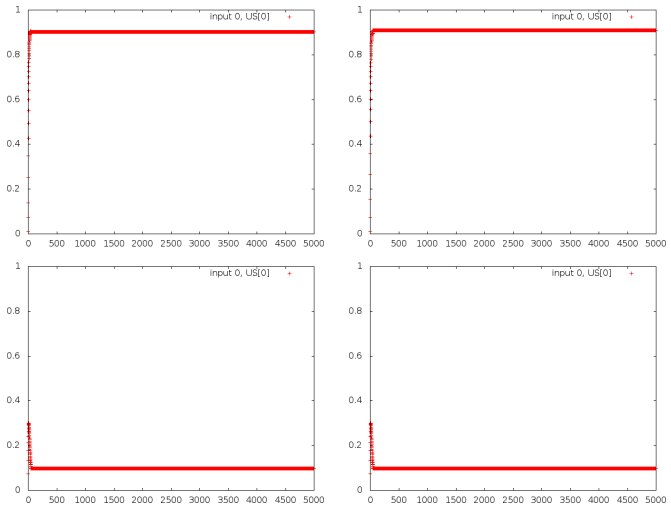


Figure 29: Outputs of Brain HA for the simple XOR
Top row: A and B (US 1), low row: AB and Z (US 0)

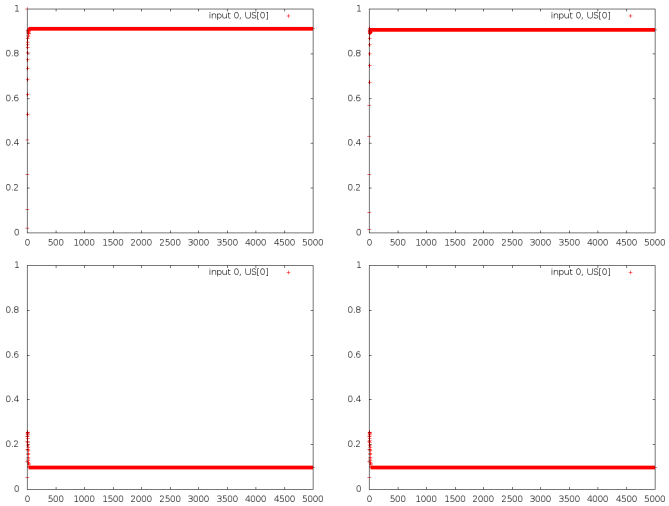


Figure 30: Outputs of Brain CHA for the simple XOR
Top row: A and B (US 1), low row: AB and Z (US 0)

And now the results sorted by ascendent φ, first the results for Brain HA:

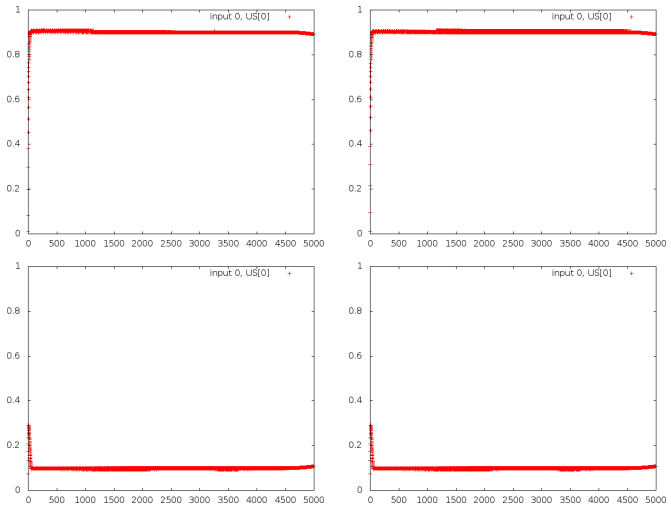


Figure 31: Outputs of Brain HA for the simple XOR, with a forgetting rate of 0.00002, not enough to show more than a little noise.
Top row: A and B (US 1), low row: AB and Z (US 0)

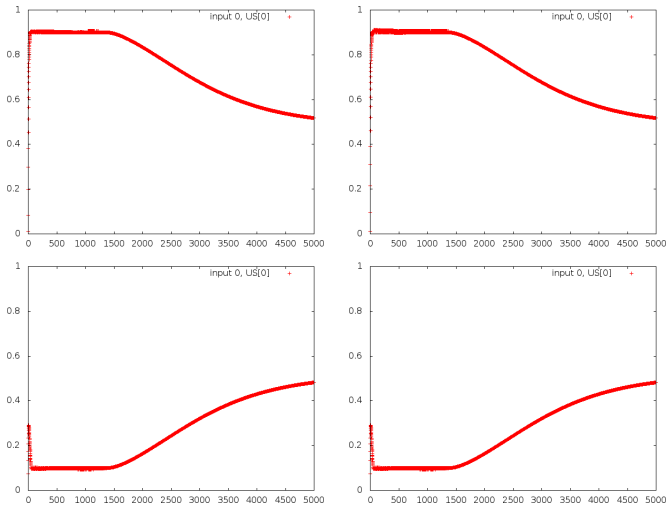


Figure 32: Outputs of Brain HA for the simple XOR, with a forgetting rate of 0.00005, which produces a gentle return to the *flatline* of the predictions.
Top row: A and B (US 1), low row: AB and Z (US 0)

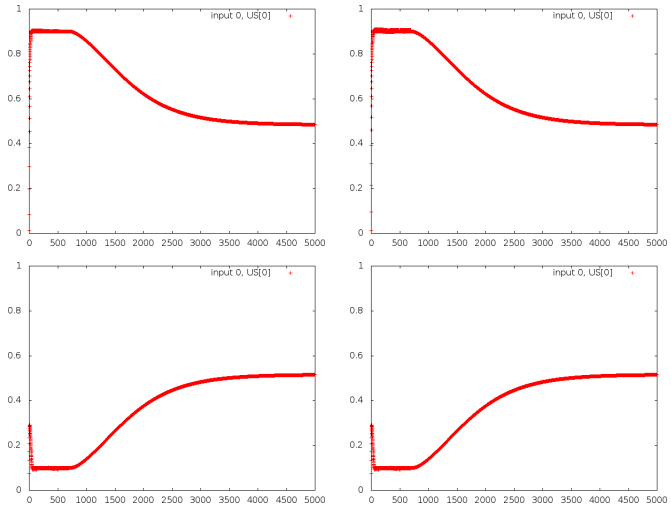


Figure 33: Outputs of Brain HA for the simple XOR, with a forgetting rate of 0.00008, which produces a faster return to predictions around 0.5, still useful.
Top row: A and B (US 1), low row: AB and Z (US 0)

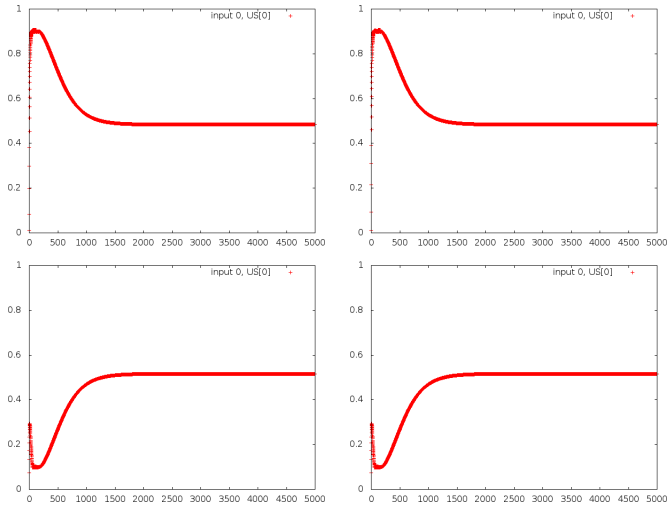


Figure 34: Outputs of Brain HA for the simple XOR, with a forgetting rate of 0.0002, which makes the predictions fall too fast, it could be said that there's no convergence.
Top row: A and B (US 1), low row: AB and Z (US 0)

And now for Brain CHA:

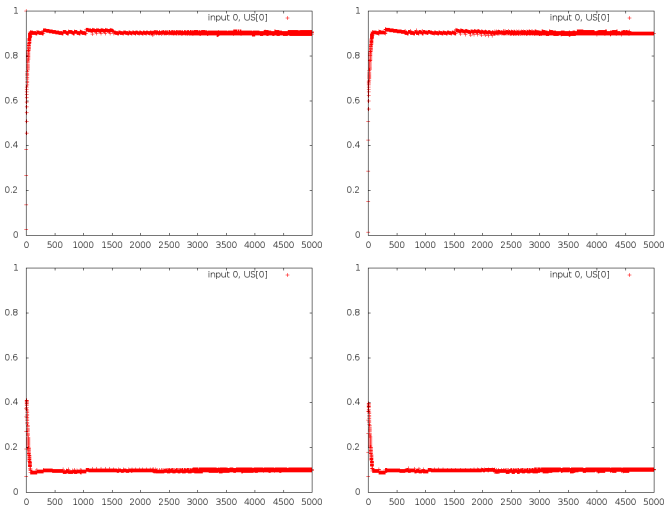


Figure 35: Outputs of Brain CHA for the simple XOR, with a forgetting rate of 0.00002
Top row: A and B (US 1), low row: AB and Z (US 0)

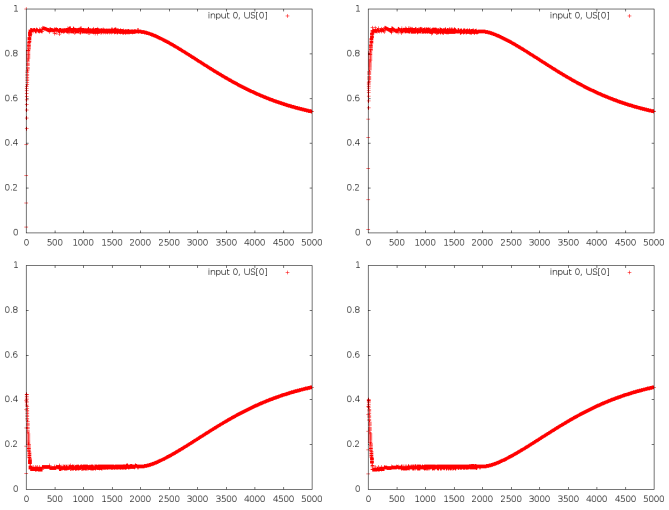


Figure 36: Outputs of Brain CHA for the simple XOR, with a forgetting rate of 0.00005
Top row: A and B (US 1), low row: AB and Z (US 0)

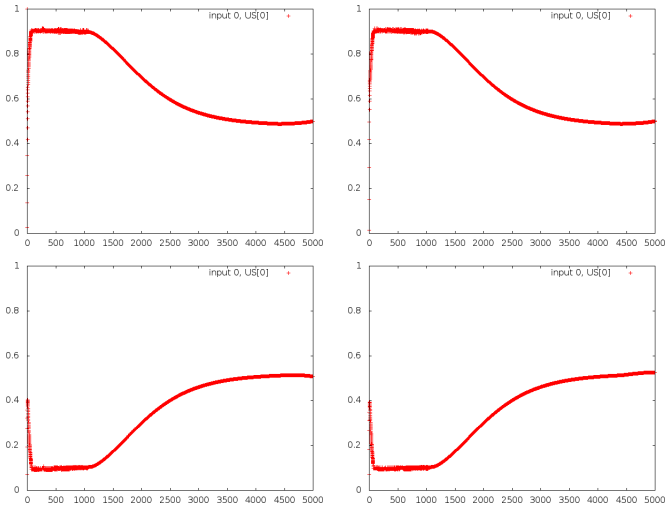


Figure 37: Outputs of Brain CHA for the simple XOR, with a forgetting rate of 0.00008
Top row: A and B (US 1), low row: AB and Z (US 0)

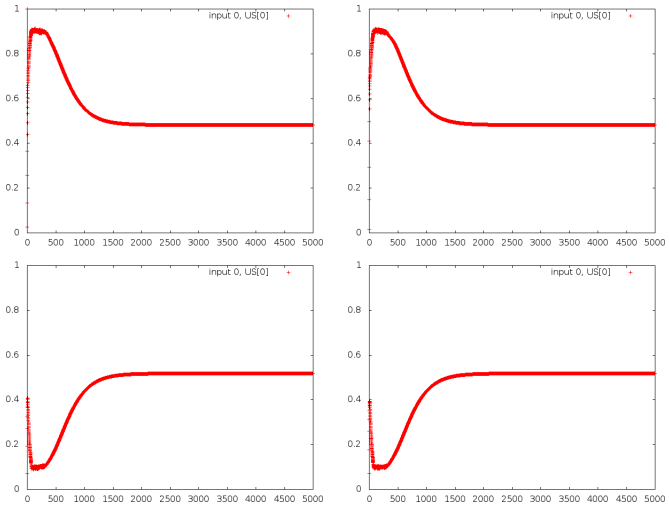


Figure 38: Outputs of Brain CHA for the simple XOR, with a forgetting rate of 0.0002
Top row: A and B (US 1), low row: AB and Z (US 0)

All four of these graphs are similar to the earlier four (for Brain HA), but in all cases the convergence is somehow noisier and lasts a few iterations more before decaying.

4.5 Discussion

Is clear that the forgetting rate φ, affects the behaviour of the `Hippocampus`. The behaviour we'd like from this mechanic is that of the figures 32 and 33, where the system gives a good prediction at the beginning of the simulation, and then this prediction starts to weaken where the `Cortex` would start to converge to a solution. With a lower rate there's no forgetting (fig. 31) and a higher rate drops the solution almost immediately (fig. 34).

On the brain that does have an `Cortex`, the output of this component delays the beginning of the downslope of the predictions, and interferes with the predictions before this slope starts. However, even though the `Cortex` will solve the problem at some point, the output from the `Hippocampus` dominates the behaviour of the `Amygdala` so the predictions flat towards 0.5, the "flatline" prediction. We believe that this domination comes from the fact that the `Hippocampus` output keeps changing, and the `Amygdala` is more affected by rapidly changing information that the slower change of the `Cortex`'s output. This dominations gives another reason to use something like the `Splitted Amygdala`.

There is no guarantee that these forgetting rates are useful for other problems, but that might be a moot point since this is only a part of the model. Maybe the "correct" φ_0 is lower than 0.00005 and modulation with signals from the rest of the model should be the main factor on forgetting. Maybe forgetting is not even needed at all. This is only a proof of concept of a part of the model, and there's lots of further work on this area, which we'll discuss on the next section.

5 Splitted Amygdala

5.1 Introduction

In the real brain, the *Hippocampus* is used to learn new and important things fast, then gradually teach them to the *Cortex*, and (maybe) gradually forget those things it has already taught. The output from both the *Hippocampus* and the *Cortex* is then used by the *Amygdala* to make a prediction based on those inputs. Our model's `Amygdala`, receives the output of both the `Cortex` and the `Hippocampus` to make that prediction. When the prediction "fails", we know that both the `Hippocampus` and the `Cortex` are failing to help the `Amygdala`. But when the prediction is successful, we don't know if any of them is still failing. In practice, what'll happen is that, since the `Hippocampus` learns faster than the `Cortex`, by design, its output will mask the `Cortex`'s. Without that information we can't dynamically adjust the learning and forgetting rates we defined and implemented at the end of section 4 as a function of the success of the predictions based only on the `Cortex` output.

We decided to create a new component that can take the `CS` and the outputs from both the `Cortex` and the `Hippocampus`, make a prediction and also differentiate between the predictions generated only from either of those components. This idea isn't just a computational patch for a problem. In the real brain, the activity from the *Cortex* arrives earlier than that of the *Hippocampus*, and the *Amygdala* is always working, so two different predictions are computed in parallel. Our implementation, however is strictly sequential: the `Cortex` gives an output, then the `Hippocampus` processes that, and then both outputs are used by the `Amygdala` to give the final prediction. Instead of introducing threading, which is never an easy task, we created a new component: the `Splitted Amygdala`.

As a black box, the `Splitted Amygdala` will take four inputs: the non-learned inputs, the output from the `Cortex` and the `Hippocampus` and subsequently the `US`. The outputs will be a prediction for the `US` (which is ready before the actual `US` arrives), and two error signals, i.e. the absolute error between the `US` and the predictions due to the `Cortex` and the `Hippocampus`.

5.2 Architecture

Internally we have two amygdalas, one takes only the output from the `Cortex` and the other only the output from the `Hippocampus`. Then we have a *Decider* component that selects which one of their predictions will be the prediction of the `Splitted Amygdala`. We tried two different approaches to this component, which will be described in the following section.

The output from the `Cortex` and `Hippocampus` are sent as inputs to the three components (two `Amygdala` and the `Decider`) and the operations for processing and learning relayed as necessary. The predictions from the two `Amygdalas` are used to calculate the errors once the actual US arrives and by the `Decider` to choose which one is a "better" prediction of the US, before it arrives.

5.2.1 Deciders

The idea of the `Decider` is to make a decision based on the previous behaviour of both `Amygdalas`. The original idea was making statistics with regards to the reliability of the predictions of each one, which we'll call *Scores*.

Scores When the US arrives, we can compute, for each `Amygdala` and for each bit of the US, the error δ of the prediction as $\delta = |P_i - US_i|$, from these values, (δ_C and δ_H) we can keep average errors[6] ($\bar{\delta}_C$ and $\bar{\delta}_H$), and from these average errors, we can compute normalized *scores* S_C and S_H as

$$S_C = \frac{Score_C}{Score_C + Score_H}$$

and

$$S_H = \frac{Score_H}{Score_C + Score_H}$$

where

$$Score_C = |\delta_C - \bar{\delta}_C|$$

and

$$Score_H = |\delta_H - \bar{\delta}_H|$$

Averaging these scores[7], we have a measure of how reliable are the predictions from each `Amygdala` for each bit of the US.

Initially we made the decision of which prediction to use based only on this score, but during testing we found that the `Amygdala` that worked with the `Cortex` produced more stable results than the one that worked with the `Hippocampus`, i.e., with better scores, even when the absolute and the average errors were lower for the latter `Amygdala` during most of the process (until the `Cortex` output moves the `Amygdala`'s predictions and it converges to a better solution). Likely these betters scores are related to the lower learning rates on the `Cortex`, which makes its outputs more stable. We then tried to make a decision based on both the score and the actual error (multiplying them), and found that it's not too hard to find examples where even this will produce erroneous predictions, specially at the beginning of the process.

Instead of searching for a better performing but more complex function of the scores and errors to make the decision we discarded this idea and implemented a different decider. The implementation of the calculation of scores is still there, but with optional compilation, and by default disabled.

Decider Amygdala The `Decider Amygdala` is (obviously) an `Amygdala` that takes the non-learned inputs (CS) and learns to predict the errors δ_C and δ_H, with a faster learning rate than the one used in the other two `Amygdalas` (which both use the same rate). The decision between both predictions is then based on the predicted error, the one that's expected to be closer to the correct answer wins. It's a simpler and more robust solution.

We found a different implementation issue here, related to the `Amygdala`'s behaviour when the inputs are static over time, discussed on 5.6

5.3 Implementation details

The interface of the `Splitted Amygdala` is as close as possible to the interface of the `Amygdala`, but no closer. The `Amygdala` takes an input vector for processing (which can be the CS, the

[6] Average over a window, not over the whole process

[7] Again, over a window, not necessarily the same than the one used for $\bar{\delta}$

`Cortex`'s output, the `Hippocampus`'s, all together, etc.), outputs a prediction and then takes another vector to learn from (the US). The `Splitted Amygdala` is less generic, since it takes takes three different inputs that it knows are the CS, the `Cortex`'s output and the `Hippocampus`'s plus the US for learning and outputs a prediction plus two error signals. Therefore, this new component is not a drop-in replacement for the `Amygdala`, and a new brain, `BrainCHAS` had to be implemented. It copies most of the behaviour of `BrainCHA`, with the necessary modifications to send the different outputs to the `Splitted Amygdala`. The methods `process()`, `learn()` and `lastOutput()` remain on the interface, with the same semantics[8]

Since this component is not a drop-in replacement for the `Amygdala`, we created a new brain, **BrainCHAS**, that implements the information transfer between all the components.

5.3.1 Fine tuning

To get these results we adjusted two of the system's knob trying to balance how fast the system processes, i.e. converges and stabilizes on a solution; with the "smoothness" of that convergence. If the learning rates are too high, the learning on the `Amygdalas` appear as sudden jumps instead of transitions. If the rates are too low, the process will take too long.

The other knob is the randomness on the starting weights of the `Cortex`'s neurons. These are selected at random around a given central weight, with a configured delta. This randomness factor impacts the variance of the iterations it takes the system to converge. If this delta is too high, then the convergence times can vary by thousands of iterations, but the delta can't be 0 since that wouldn't be a good model of the *Cortex*.

For these tests we settled on these values:

- **Learning rate for the `Amygdalas`:** 0.005
- **Learning rate for the `Decider Amygdala`:** 0.03
- **Learning rate for the `Cortex`:** 0.002
- **Deltas for the weights of the `Cortex`'s neurons:** 0.01

The testing protocols for this portion of the research are described on the section 5.4, the results are on 5.5 and discussed on 5.6

5.4 Testing scenarios

With the `Splitted Amygdala` we wanted a scenario where the `Hippocampus` worked better for some inputs and the `Cortex` better for others. What we decided to try is a version of the Double XOR with a different `Cortex`. Instead of having a single input map we used two, each one of four units and connected to a middle map of two units. This two middle units are both connected to the single unit output map. This scenario is called **Double mapped XOR**, and was tested presenting all stimuli in all iterations, without any variation.

[8] Although for some reason the latter method is called `output()`

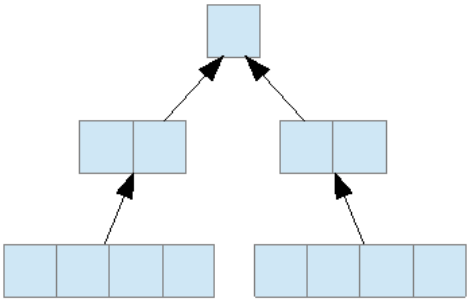


Figure 39: `Cortex` used to run the Double mapped XOR scenario

The inputs are a XOR in one of the maps and either 0s or 1s on the other one, so we have:

Set	Input	Left map	Right map	US
XOR 0	A	[1 0 0 1]	[0 0 0 0]	[1]
	B	[0 1 1 0]	[0 0 0 0]	[1]
	AB	[1 0 1 0]	[0 0 0 0]	[0]
	Z	[0 1 0 1]	[0 0 0 0]	[0]
XOR 1	A	[1 0 0 1]	[1 1 1 1]	[1]
	B	[0 1 1 0]	[1 1 1 1]	[1]
	AB	[1 0 1 0]	[1 1 1 1]	[0]
	Z	[0 1 0 1]	[1 1 1 1]	[0]
0 XOR	A	[0 0 0 0]	[1 0 0 1]	[1]
	B	[0 0 0 0]	[0 1 1 0]	[1]
	AB	[0 0 0 0]	[1 0 1 0]	[0]
	Z	[0 0 0 0]	[0 1 0 1]	[0]
1 XOR	A	[1 1 1 1]	[1 0 0 1]	[1]
	B	[1 1 1 1]	[0 1 1 0]	[1]
	AB	[1 1 1 1]	[1 0 1 0]	[0]
	Z	[1 1 1 1]	[0 1 0 1]	[0]

A variation on this scenario, called **Double mapped crossed XOR**, was briefly tested, but the `Cortex` described can't solve any of these inputs. In this scenario half of the XOR is in the left side map and the other half on the right side one. For example:

Set	Input	Left map	Right map	US
XOR 0 / XOR 0	A	[1 0 0 0]	[0 1 0 0]	[1]
	B	[0 1 0 0]	[1 0 0 0]	[1]
	AB	[1 0 0 0]	[1 0 0 0]	[0]
	Z	[0 1 0 0]	[0 1 0 0]	[0]

5.5 Results

5.5.1 Simple XOR

For the `Splitted Amygdala` we show the output of the two inner `Amygdalas`, the `Decider Amygdala` and the chosen predictions. First, as a base case to show that the idea works, we ran the Simple XOR on it.

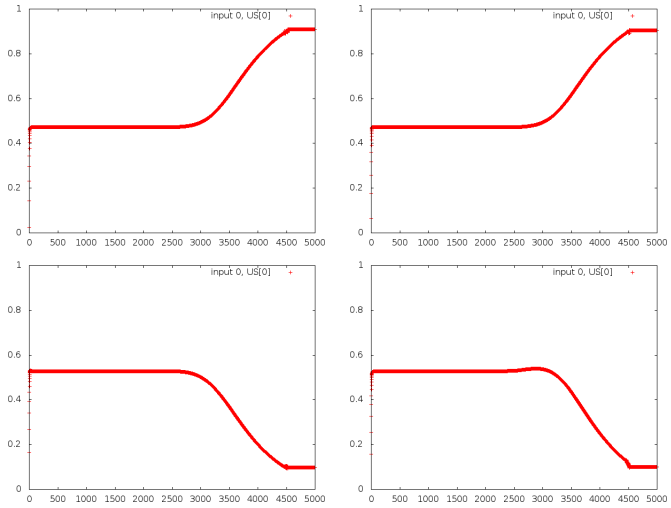


Figure 40: Predictions by the `Cortex`'s `Amygdala`, very similar to figure 2.
Top row: A and B (US 1), low row: AB and Z (US 0)

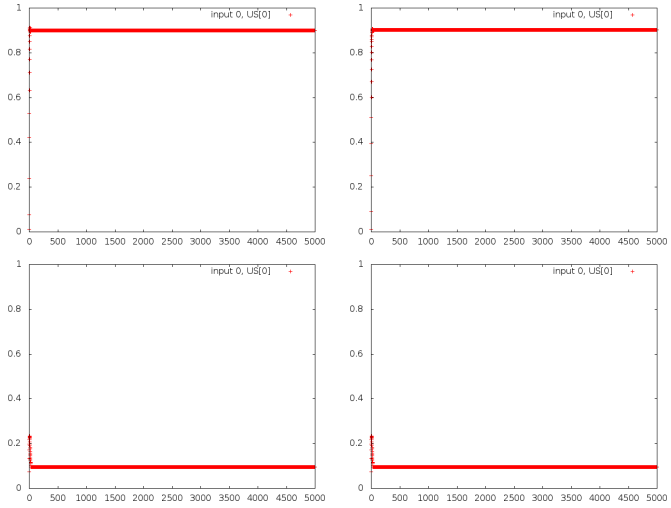


Figure 41: Predictions by the `Hippocampus`'s `Amygdala`, very similar to figure 3.
Top row: A and B (US 1), low row: AB and Z (US 0)

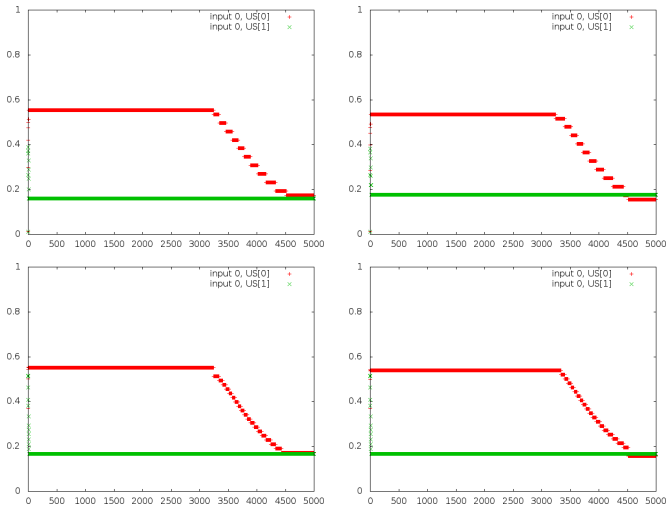


Figure 42: `Decider Amygdala` predictions, on red the predicted error for for the `Cortex`'s `Amygdala`, on green, the predicted error for the `Hippocampus`' `Amygdala`.
Top row: A and B (US 1), low row: AB and Z (US 0)

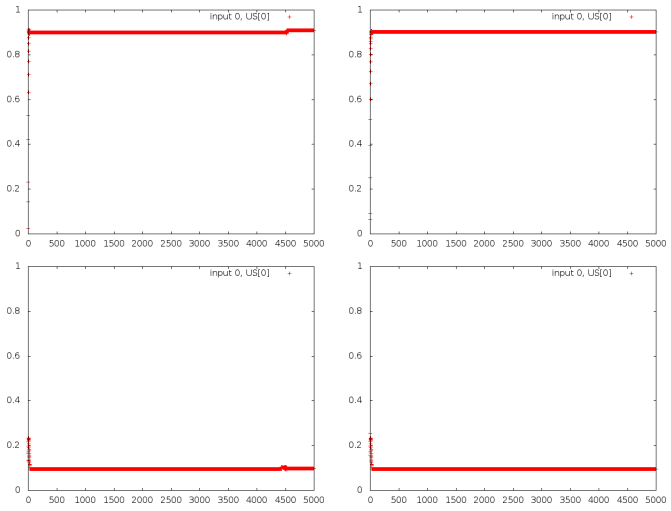


Figure 43: `Splitted Amygdala` predictions: the output selected based on the `Decider Amygdala`'s predictions. the output selected based on the `Decider Amygdala`'s predictions. The only difference is for A, where the output changes for the last few hundred iterations.
Top row: A and B (US 1), low row: AB and Z (US 0)

5.5.2 Double map XOR

Then, as explained before, we ran the double map scenario. Its important to remember that all 16 different stimuli are presented to the system together, as part of a single simulation. To simplify the presentations we'll present them in groups of 4, as subproblems, but these aren't independent.

XOR - 0 XOR on the left hand map, 0s on the right one:

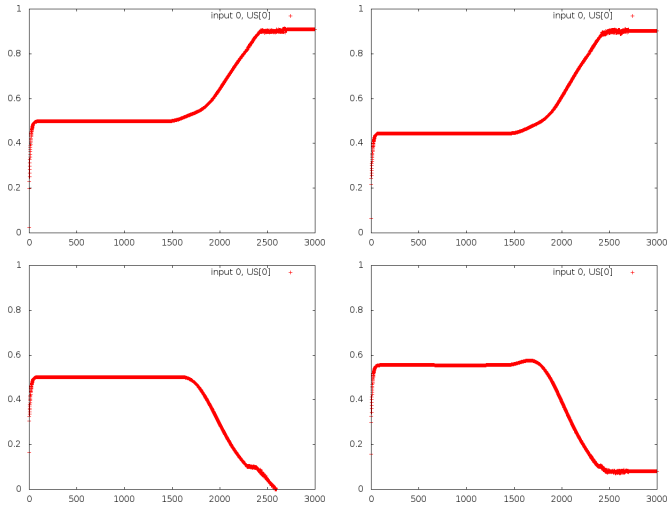


Figure 44: Predictions by the `Cortex`'s `Amygdala`. For AB the predictions go below 0, likely converging near -0.1. Otherwise the graphs are similar to figure 2.
Top row: A and B (US 1), low row: AB and Z (US 0)

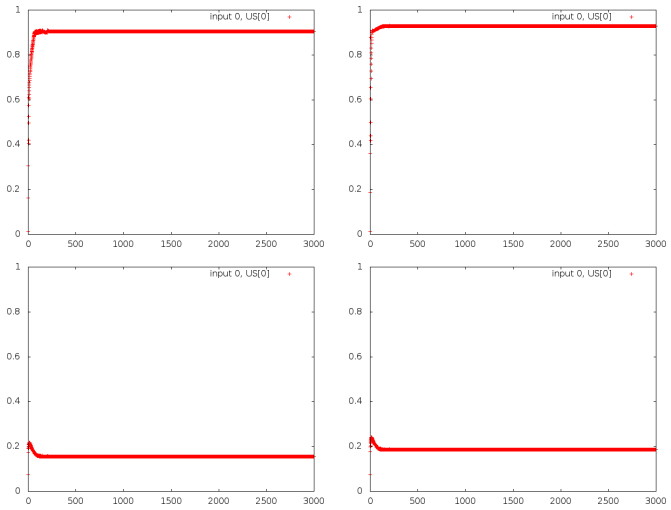


Figure 45: Predictions by the `Hippocampus`'s `Amygdala`, similar to 3.
Top row: A and B (US 1), low row: AB and Z (US 0)

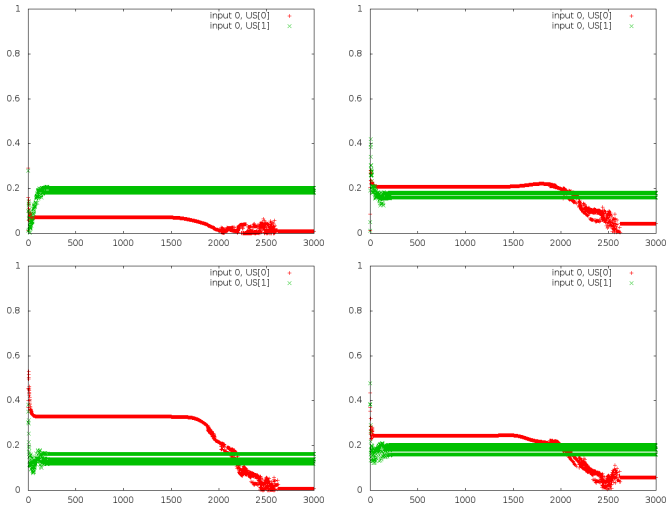


Figure 46: `Decider Amygdala` predictions, on red the predicted error for for the `Cortex`'s `Amygdala`, on green, the predicted error for the `Hippocampus`' `Amygdala`. For some of the inputs it fails to correctly predict the error of the `Cortex`'s, by a large margin in the case of A. For the `Hippocampus`' it cycles around a few values.
Top row: A and B (US 1), low row: AB and Z (US 0)

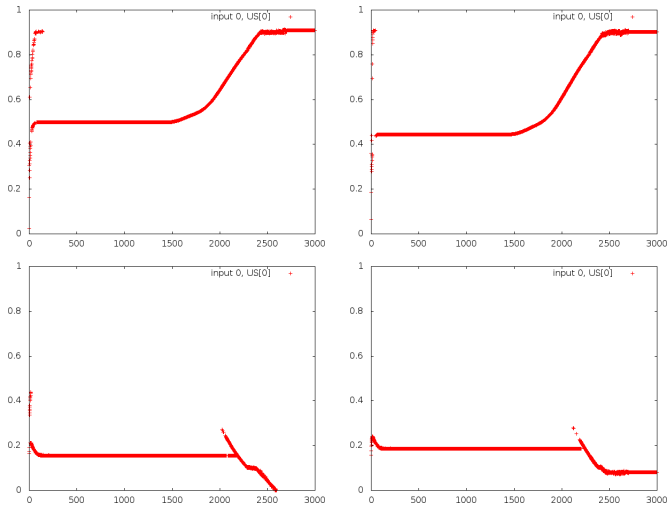


Figure 47: `Splitted Amygdala` predictions: the output selected based on the `Decider Amygdala`'s predictions. Due to the failure on the predictions for A and B, the component chooses the `Cortex`'s `Amygdala` prediction, even when the `Hippocampus`'s is better. Also, since for AB the predicted error for the `Hippocampus`' `Amygdala` cycles, there's a period where the output oscilates between both predictions.
Top row: A and B (US 1), low row: AB and Z (US 0)

XOR - 1 XOR on the left hand map, 1s on the right one:

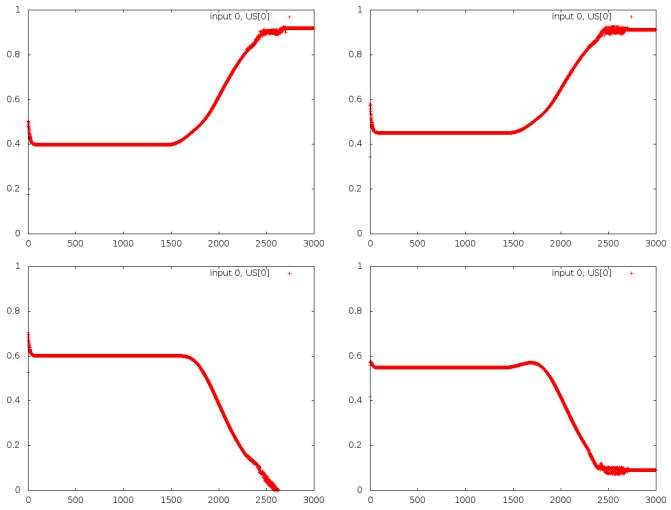


Figure 48: Predictions by the `Cortex`'s `Amygdala`. Very similar to figure **??**.
Top row: A and B (US 1), low row: AB and Z (US 0)

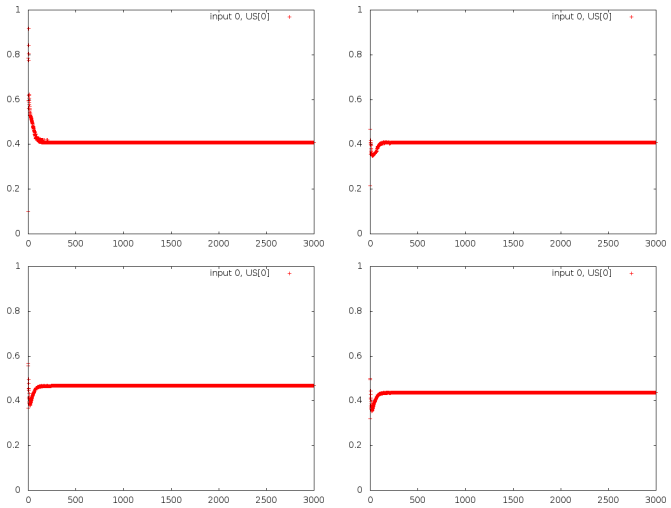


Figure 49: Predictions by the `Hippocampus`'s `Amygdala`, which apparently can't solve this subproblem so it *flatlines*.
Top row: A and B (US 1), low row: AB and Z (US 0)

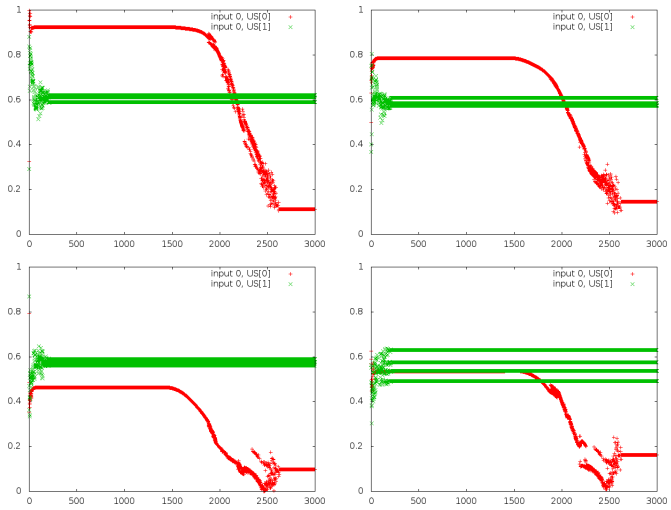


Figure 50: `Decider Amygdala` predictions, on red the predicted error for for the `Cortex`'s `Amygdala`, on green, the predicted error for the `Hippocampus`' `Amygdala`. Again, large errors for A and B on red, but here the `Decider Amygdala` predicts worst behaviour than the real. Also, for Z the cycle of predicted values for the `Hippocampus`' `Amygdala`'s error is wider.
Top row: A and B (US 1), low row: AB and Z (US 0)

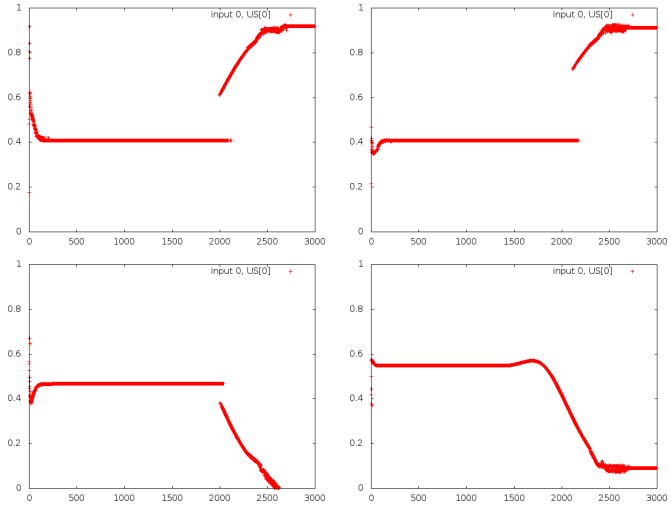


Figure 51: `Splitted Amygdala` predictions: the output selected based on the `Decider Amygdala`'s predictions. Again, since the predictions are wrong, the selections are too, only this time it's picking the output from the `Hippocampus` instead of the `Cortex`.
Top row: A and B (US 1), low row: AB and Z (US 0)

0 - XOR XOR on the right hand map, 0s on the left one:

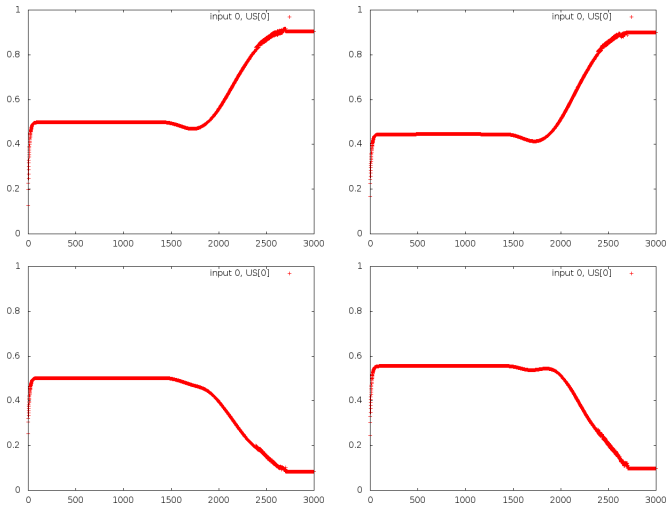


Figure 52: Predictions by the `Cortex`'s `Amygdala`. Also very similar to figure **??**.
Top row: A and B (US 1), low row: AB and Z (US 0)

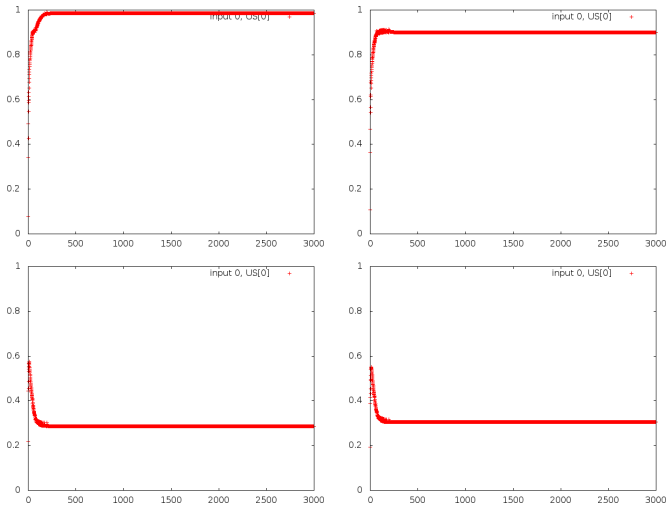


Figure 53: Predictions by the `Hippocampus`'s `Amygdala`. Similar to figure 4, but with a better prediction for A.
Top row: A and B (US 1), low row: AB and Z (US 0)

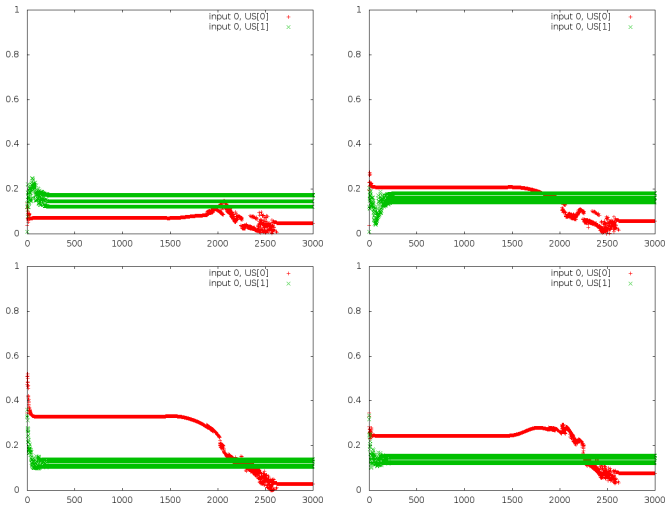


Figure 54: `Decider Amygdala` predictions, on red the predicted error for for the `Cortex`'s `Amygdala`, on green, the predicted error for the `Hippocampus`' `Amygdala`. Similar problems than on figure 46
Top row: A and B (US 1), low row: AB and Z (US 0)

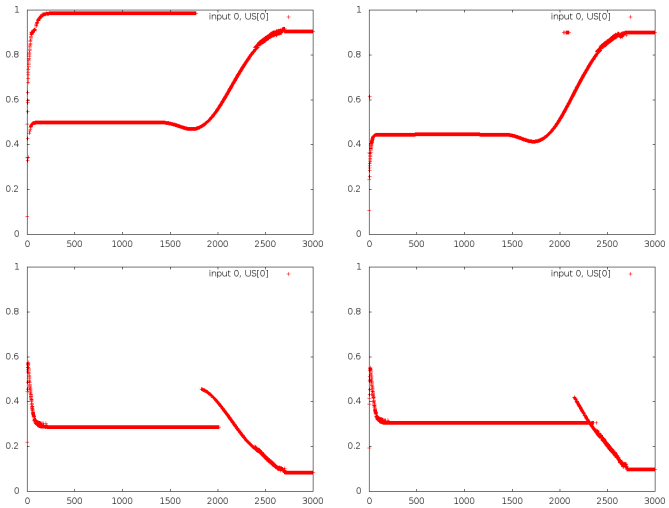


Figure 55: `Splitted Amygdala` predictions: the output selected based on the `Decider Amygdala`'s predictions. Something very strange happens for A, it doesn't seem to be a product of the previous figure, unless the cycle of predicted errors for the `Hippocampus`' `Amygdala` has a value below 0 which wouldn't appear on the graph but closer to 0, which would be better than the predicted error for the `Cortex`'s `Amygdala`.
Top row: A and B (US 1), low row: AB and Z (US 0)

1 - XOR XOR on the right hand map, 1s on the left one:

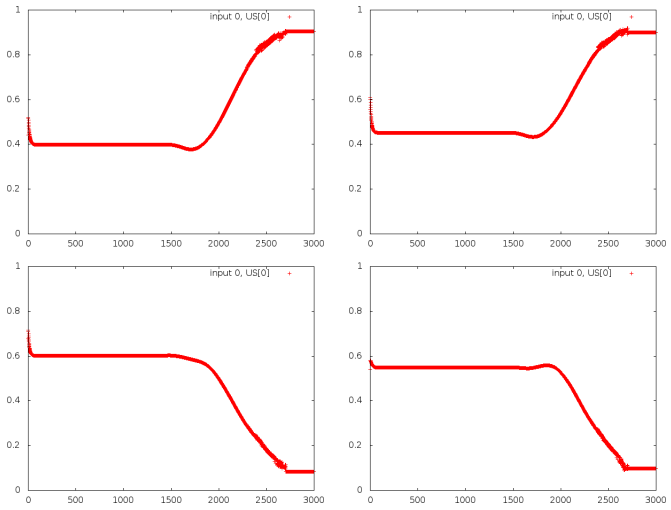


Figure 56: Predictions by the `Cortex`'s `Amygdala`. Very similar to figure **??** as well.
Top row: A and B (US 1), low row: AB and Z (US 0)

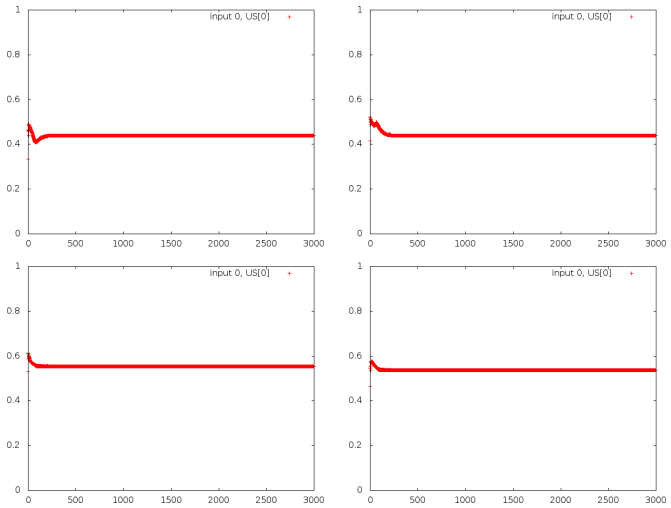


Figure 57: Predictions by the `Hippocampus`'s `Amygdala`. As for the XOR - 1, this can't be solved by this part of the system.
Top row: A and B (US 1), low row: AB and Z (US 0)

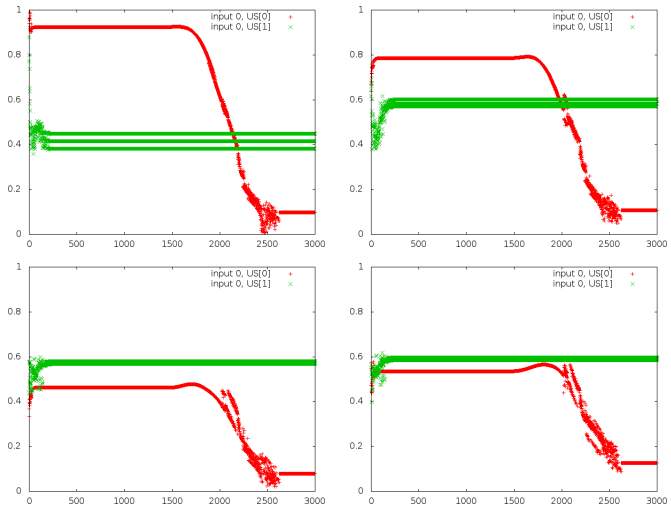


Figure 58: `Decider Amygdala` predictions, on red the predicted error for for the `Cortex`'s `Amygdala`, on green, the predicted error for the `Hippocampus`' `Amygdala`. Generally the same problems than those shown in the figure 50
Top row: A and B (US 1), low row: AB and Z (US 0)

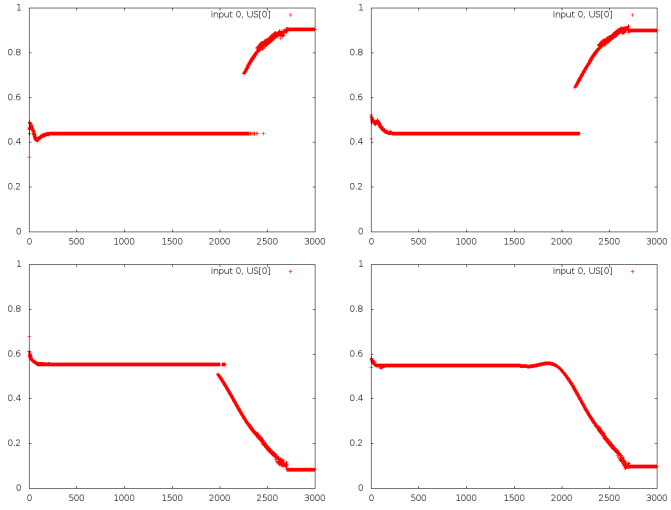


Figure 59: `Splitted Amygdala` predictions: the output selected based on the `Decider Amygdala`'s predictions. Again, same problems than those shown in the figure 51
Top row: A and B (US 1), low row: AB and Z (US 0)

5.6 Discussion

5.6.1 Simple XOR

The first tests show that the idea seems to work. On this use case, the `Hippocampus` gives a pretty good answer, and the `Cortex` only gets better on a few occasions once it converges, and the prediction generated by the `Splitted Amygdala` only changes then (which wasn't happening with the scoring system).

Of special note are the jumps or staircase that appear on the `Decider Amygdala`'s graph. These are due to the higher learning rate that the `Decider Amygdala` uses, which induces these jumps instead of the smoother transitions seen on the other graphs.

5.6.2 Double Map XOR

With this scenario we get more interesting results. On one hand, setting the "unused" map to ones makes `Amygdala` assigned to the `Hippocampus`' output predicts useless values, which is what we designed this part of the scenario for. This would have been an example of a subproblem where the `Splitted Amygdala` would choose the `Cortex` + `Amygdala`'s output. Notice that since this is a subproblem, i.e. it runs alongside the rest of the examples which are correctly solved by `Hippocampus` + `Amygdala`, we cannot say whether it is the `Hippocampus` that's overwhelmed by so many 1s and outputs useless information, or if it's the `Amygdala` that doesn't know what to do with this information. We also don't know if in the former case the problem is on `CA3` or `CA1`.

On the other hand, the graphs showing the behaviour of the `Decider Amygdala` ilustrate the next problem of this component. Figures like 46 and 47, show that the decider often makes bad predictions for the error of the `Cortex` + `Amygdala` and stays there until the `Cortex` starts affecting the `Amygdala`'s predictions. But before that the `Splitted Amygdala` will have chosen the `Cortex`'s prediction as its final prediction, which is wrong. The opposite problem can happen too, like on the XOR - 1 subproblem.

5.6.3 Problems with the `Amygdala`

To explain the problem, we need to explain with some detail how the `Amygdala` learns. On each iteration of a problem, each neuron of the amygdala will learn (i.e. have its weight W modified) according to:

$$W_{ij} = W_{i(j-1)} + \lambda \times I_i \times \delta_j$$

where

i is the position on the input and the neuron number.

j is the stimulus number ($0 \leq j <\leq$ # stimuli on the problem).

λ is the learning rate.

I is the input vector, i.e. the stimulus.

δ_j is the error: $US_j - prediction_j$

I_i can be either 0 or 1. In the first case, clearly, W remains unchanged. Since λ is a constant, we can define $k = \lambda \times I_i$. Now, for a complete iteration of the problem, for a given neuron (i.e., fixing i), calling the starting weight W_s and considering that that particular neuron will be affected by n stimuli (i.e., there are n inputs for which $I_i = 1$), we'll have:

$$W_0 = W_s + k \times \delta_0$$
$$W_1 = W_0 + k \times \delta_1$$
$$W_2 = W_1 + k \times \delta_2$$
$$\dots$$
$$W_n = W_{n-1} + k \times \delta_n$$

Replacing one equation on the following:

$$W_0 = W_s + k \times \delta_0$$
$$W_1 = W_s + k \times \delta_0 + k \times \delta_1$$
$$W_2 = W_s + k \times \delta_0 + k \times \delta_1 + k \times \delta_2$$
$$\dots$$
$$W_n = W_s + k \times \delta_0 + k \times \delta_1 + \dots + k \times \delta_{n-1} + k \times \delta_n$$

So finally:

$$W_n = W_s + k \left(\sum_{j=0}^{n} \delta_j \right)$$

And when the amygdala reaches a stable state,

$$\sum_{j=0}^{n} \delta_j \approx 0, \text{ so } W_n = W_s$$

This is both a bug and a feature. It's a feature because it means that yes, the amygdala does reach a stable state. It's a bug because that stability does not depend on each $|\delta_j|$, it's not affected by how large the error of the prediction is. This is why the "flatline" of the `Amygdala` is around 0.5.

The actual problem is the use we are giving this neural network. A different one might be better for the role of Decider. So far we were using the `Amygdalas` with two inputs: on one side, the activities used as input, whether from the `Cortex`, the `Hippocampus`, both, etc. and on the other side, the actual US, once it arrives. These inputs came from the sensorial input, the `Cortex` and the `Hippocampus`. The first one is constant (for each stimulus). The input from the

`Cortex` varies over time as it learns and so the input to the amygdala varies. The Hippocampus includes the Cortex's input, so it varies too.

The new examples tested on this period use these inputs differently: For the decider amygdala, the inputs are constant, what changes is the "US", in this case, the actual predictions from the `Cortex` and the `Hippocampus`' amygdala. For the `Cortex` +`Amygdala`'s error prediction it converges to a "wrong" result early on and then it stays there until the `Cortex` starts converging and the prediction changes.

6 Further work

There are many ideas hanging from this work, regarding further development on the `Hippocampus` itself, on combining it with the rest of the model and improving testing.

6.1 Dynamic forgetting

Although we implemented small unit tests for the dynamic model of forgetting, we haven't tested it together with the rest of the system and simulations of scenarios. To fully integrate this `Hippocampus` with the other components we needed to implement two features.

One is a measure of the errors commited by the `Cortex`, which we now have as a signal from the `Splitted Amygdala`. We could use this signal as part of the strength of the memories stored on the `Hippocampus`, and/or to decide if we can start forgetting or at least increasing the forgetting rate for a stimulus.

6.2 Consolidation

The other feature is the process called *Consolidation*. This is a process by which the *Hippocampus* "teaches" the *Cortex* what the former has stored, to help the latter create its representations. There are many reports of how this system might work from a biological perspective, but no computational models. We believe that a model for this process, no matter how rudimentary, is needed to complete the information loop between the `Cortex`, the `Hippocampus` and the `Amygdala` and control how and when the information can be forgotten. Every time we discussed the more advanced features of forgetting we came back to the interaction between the `Cortex` and the `Hippocampus` which right now is one way (the ouput of the `Cortex` feeds the `Hippocampus` via the `ERC`) and consolidation is the other, still missing way. Right now, without consolidation the `Hippocampus` would be forgetting inputs hoping that the `Cortex` is learning them, and taking no active part on that process.

The problem of modeling consolidation is very complex, but we believe that implementing the simplest model we can think of and start running simulations to see what we are missing can be a successful first step. We propose implementing a "sleep period" feature on the brains and scenarios, since most of the biological data indicates that consolidation "runs" during periods of low to no activity like sleep, and on those periods feed the contents of the `Hippocampus` back to the `Cortex`, to train it with the data stored and not the outside stimuli. The `random_recall()` feature (see section A.2) can be a good enough starting point for "selecting" which inputs to send, so the only problem remaining is exactly how to feed it back to the `Cortex`, since what it sends to the `Hippocampus` is *all* of its activity and right now there's no way to input that activity back.

6.3 Decider

This is an obvious dangling thread, we haven't found yet the right decider. We need something similar to the amygdala, but that cares more about the errors of it's predictions to avoid the current behaviour where it settles on a prediction too far from reality, and stays there until the US (the predictions from the other `Amygdalas`) change.

6.4 Binary vs. Real Computation

As mentioned many times, the current system takes binary inputs and tries to generate binary outputs. This is what we call binary computation, as opposed to real computation, where the inputs are real (`double`) numbers and so are the outputs. The difficulty with this mode of computation is more semantic than syntactic, since the system internally works with `doubles`. In fact the inputs to `Hippocampus` tend to be real numbers since they arrive while they consolidate to 0 or 1 on the `Cortex`.

The biggest problem is with separation/completion: it's easy to distinguish `[1 0 0 1]` from `[0 1 0 1]` but it's harder to distinguish `[0.8 0 0 1]` from `[0.3 0 0 1]`. Right now we can't decide whether the second input is a different one or the first one that hasn't finished consolidating. A similar problem can be found with forgetting: a value of 0.42, is a recall of a previous storage of that value or is it a different value that is being forgotten?

When we considered these problems we decided that using real numbers was too much of a problem without adding benefits and, if it came to that, we could always modify `DG` to convert real numbers to a binary representation.

Other points of the system that expect the inputs to be between 0 and 1:

- The `Amygdala`'s neurons cap their outputs to that range.
- The values stored on `CA3` are also capped to 1. Additionaly, when the value is input for the first time is set to $1 \times {}^{s}/_{\tau}$ (where s is the strength of the memory (1 by default) and τ is the time constant).
- Forgetting is capped to 0.
- `random_recall()` (see section A.2) assumes that `CA3` tries to store 1s.

6.5 Continuous use

All of our simulations are restricted on time and scope. We run the system for a number of iterations and for a certain number of stimuli. Another line of investigation involves releasing the system on the wild, or at least on Minecraft, and letting it act and react on the environment. The question then is, what problems could this new operation mode cause on the system.

The first problem we see is that right now it takes too long for the `Cortex` to learn the associations it needs, due to the low learning rates. Increasing this rates can cause sudden "jumps" on the predictions instead of smooth(er) transitions from one answer to another. So if a fast reaction is needed, the `Hippocampus` becomes essential.

With this component the representation of the inputs becomes particularly important, they have to be sparse. The scenario "Double mapped XOR" used to test the `Splitted Amygdala` (section 5.4, results on 5.5) shows that too many inputs set to one overwhelm the component and its outputs become useless to the `Amygdala`. Whatever representation is chosen to interpret the environment, it has to be careful with this, but it should also be careful with the size of the inputs, because that may exceed the capacity of the machine in both memory and processing speed.

Then there's the issue of capacity of the `Hippocampus`, how many different inputs can be stored until they start to overlap and generate unexpected results. As stated on section **??**, we

didn't get conclusive results for that problem, so while it's clear that the number of different inputs the component can store is limited, we don't know if this is below the number of different inputs the environment can and/or will generate while the system runs. Exceeding that capacity could either be a case for the implementation of forgetting or to modify the representation and make it sparser, with the caveats already stated.

6.6 `CA1`, the weak link

As a general observation, while the model seems solid for the current and immediately foreseeable uses, we have the intuition that, when something breaks, it will probably be `CA1`. The behaviour shown during the tests for capacity and scale indicates that it's somehow fragile. The implementation looks a bit naïf and has not been stress tested. If the function used on `DG` changes, this implementation might simply not be enough to translate for it's purpose.

It's also tempting to use this component to redistribute the stored inputs from the `Hippocampus` back to the different levels of the `Cortex` during consolidation. We advice against this model, unless it has strong biological evidence to back it up, and recommend instead to move this functionality to the `ERC` or another intermetdiate component. Giving `CA1` more functionality than translating from one "language" to the other would add unnecesary complexity and probably get us nowhere.

6.7 Probes and better test tools

Some of the tests we ran didn't produce results not because the tests itself are wrong, but because we can't get enough data of the behaviour of the system. Here's a brief list of tools and outoputs that we think would be useful to improve this situation:

- Either better criteria for what's a "mistake" or
- a way to run the capacity tests with at least the brain without `Cortex` (BrainHA) and define "mistake" as the point where the predictions start to fail.
- Easier ways to look into the behaviour of the `Amygdala`, `Cortex` and `Hippocampus` than dumping the states to files or printing debug lines. Ideally these probes should be presentable in some sort of graph.
- Better scenarios to test learning by heart, where a subset of inputs appear sporadically and need to be learned by the `Hippocampus` for the brain to be "succesful"

Appendix A Other implementations

A.1 Corpus

The scenarios are described on json files which define the stimuli (inputs), the USs and their relations. Then the program creates the stimuli on runtime and presents it to the selected brain.

One problem with this is that this files define implicitly an order in which the stimuli are presented, and in these problems the order matters. We wanted to try presenting the stimuli randomly, and to do so, we needed to have all the stimuli created and mixed before running the simulation. We call this "precompiled" list of stimuli a *corpus*.

Implementation

The implementation of the corpus input has two halves, the script to create the corpus file from a scenario file (described below) and a new scenario class (`ScenarioCorpus`) to read the inputs from those files instead of creating them. We also modified the `Loader` class to check for the parameter `USE_CORPUS` on the scenario's config file and create the corresponding `Scenario` class.

`create_corpus.py` This is a (relatively) simple script written in python that takes a scenario file (and an optional `-r` parameter to enable randomizing) and creates a corpus (`corpus.txt`) with the dimensions of the problem on the first line (input size, US size) and then a stimulus on each line on the format `<input> <us> <delay>` (the latter is the amount of iterations until the US is presented, that is defined on the scenarios)

A.2 Random Recall

This is a method of the `Hippocampus` to recall a stored input. It has two forms: `random_recall()` obtains an input on a single call, while `build_random_recall()`, builds one over time (which can be obtained with the method `output()`, as usual).

Implementation

These methods of the `Hippocampus` call `random_recall()` on `CA3` to find a stored stimulus and then feed that recalled input to `CA1` to build the definitive output either over time or in the single call.

`CA3`'s `random_recall()` randomly selects cells on the storage matrix until it finds a non-zero cell (i.e., a cell with a value above a certain threshold). Once it found that W_{ij}, it walks all of the row i and the column j, and copies all of the values from both to the output. The random recall on `CA3` is always done fully on one call.

Appendix B `main_bis`

To ensure backwards compatibility with the work that was being done earlier and in paralel with this one, we used a different `main()` on `main_bis.cpp` built using `Makefile_bis`, with these modifications.

Configuration by parameter We used the library `getopt` to be able to define which brain and scenario will be simulated on run-time. Here is the help output:

```
$ ./Cortex_bis --help
./Cortex_bis ([ -b <brain_def> ] [ -c <cortex filename> ] [ -a <amygdala filename> ] [ -s <sce
              | -h )
          -b arg, --brain=arg:     Brain to use
                                    One of:  ac  (Amygdala and Cortex).
                                             ha  (Hippocampus and Amygdala).
                                             ach (All together).
          -c arg, --cortex=arg:    Cortex definition filename
          -a arg, --amygdala=arg:  Amygdala definition filename
          -i arg, --hippo=arg:     Hippocampus definition filename
          -s arg, --scenario=arg:  Scenario definition filename
```

This is implemented with a new `scenario()` function and the class `Options`

`scenario_chas` is a new method to use the BrainCHAS, since building it is different that building all of the other brains.

We strongly recommend that all this functionality is merged back with the "standard" `main.cpp`.

Appendix C Configuration files for the Scenarios

The configuration of the scenarions described in section ?? are on the directory `cortex-learning/scenarios/Learn_by_h`
Each scenario is described by a file named `Scenario*.json` and the `Cortex` and `Amygdala` used are described on the corresponding `AmygdalaConfig*.json` and `CortexConfig*.json`. The few configurable values of the `Hippocampus` are on the files `Hippiocampus.json` and `Hippocampus_forgets.json`, and are common for all scenarios.

The Simple XOR scenario (sec. 2.4) and it's variations are defined on these files:

```
Scenario_Simple-XOR-all_equal.json
Scenario-Simple_XOR-corpus.json
Scenario-Simple_XOR.json
Scenario-Simple_XOR-proba-corpus.json
Scenario-Simple_XOR-proba.json

AmygdalaConfig_Simple-XOR-SimpleCortex.json

CortexConfig_Simple-XOR-SimpleCortex.json
```

For the Double XOR (sec. 3.3) we have

```
Scenario-Double_XOR.json
Scenario-Double_XOR-proba.json

AmygdalaConfig-Double_XOR.json

CortexConfig-Double_XOR.json
```

And finally for the Double Mapped XOR and Double mapped crossed XOR (sec. 5.4),

```
Scenario-Double_map.json
Scenario-Double_map_crossed.json

AmygdalaConfig_Double_map.json

CortexConfig_Double_map.json
```

Notice that the `Splitted Amygdala` uses only the behaviour constants for the `Amygdalas` and defines "manually" the sizes of its component `Amygdalas` based on the definition of the problem to solve.

Appendix D Scripts

At `cortex-learning/plot_scripts/` there are a few scripts to quickly run and plot these examples. There's also a script to create corpuses, written in python, on the "home" directory.

D.1 Requirements

The plotting scripts use gnuplot to plot the data and imagemagick to stitch the plots for the different stimuli together into a single image. They also rely on a certain folder organization of the repository and the paths of the files where the main program outputs the predictions and other data. Some of the directories are created by the scripts themselves, and all of these directories are presetted as variables at the beginning of the scripts.

D.2 `plot_amygdala.sh`

This is the core script for plotting the output of the simulations. It was designed to plot the predictions of the `Amygdala`, hence the name, but now it's mostly used by the rest of the scripts to plot a series of values which might be predictions, errors etc.

```
$ ./plot_amygdala.sh -h
Use: ./plot_amygdala.sh <filename> <nuber of inputs> <US size> [input offset (zero based)]
                        <output file> [yrange]
  where
    filename:         Path of the file with the data to plot.
    number of inputs: Number of different inputs on the problem.
    US size:          Bits on the US.
                      A different line will be plotted for each input, for each US.
    input offset:     Use to skip inputs from the problem.
    output file:      Path of the file to write (png).
    yrange:           Top and bottom value on the plot written as [top:bottom].
                      The default is [0:1].
```

The parameters between angles (< >) are mandatory and those between brackets ([ ]) are optional. The order of the parameters is important, and not all of the combinations of optional and mandatory parameters are guaranteed to work.

D.3 Plotting scripts

There is a script for each of the problems, plus the variations for the first one:

`plots-simple-all_equal.sh`
Runs the simple XOR with all stimuli presented equally, using al three "original" brains (CA, HA and CHA) and plots the results.

`plots-simple-all_equal-corpus.sh`
Same as before, but uses a corpus randomizing the inputs.

`plots-simple-proba.sh`
This runs the same XOR, but the stimuli for AB and Z have a probability of 0.2 to appear.

`plots-simple-proba-corpus.sh`
Uses the corpus created against the scenario with the lower probability for AB and Z to present the stimuli randomized.

`plots-double-all_equal.sh`
Runs the double XOR with all stimuli presented equally, using al three "original" brains (CA, HA and CHA).

`plots-double-proba.sh`
Runs the double XOR using al three "original" brains (CA, HA and CHA), but the stimuli for the "second" XOR have a probability of 0.2 to appear.

`plots-simple-brain-chas.sh`
This runs the simple XOR against the brain with the `Splitted Amygdala`, which has a different output so the creation of the graphs is different. A few parameters on the first lines that control whether to use a corpus and whether that corpus is randomized. Notice that enabling corpus on the script doesn't enable it on the Scenario.

`plots-double_map-brain-chas.sh`
This runs the double mapped XOR against the brain with `Splitted Amygdala` (BrainCHAS). It also has the parameters to control whether to use a corpus and its randomization.

`plots-double_map_crossed-brain-chas.sh`
This runs a version of the double mapped crossed XOR against brain CHAS, again with the parameters to control corpus use and its randomization.

Appendix E Unit tests

A set of reduced and artificial tests were implemented to keep track of the functionality and the modifications of the `Hippocampus` and to easily test the new functionality. As usual, each test is a single function, and then there's a function that runs each one, and another that calls all of them and prints the results. Three sets of tests were implemented, and all of them are on the directory `test/`. There's a `main.cpp` that may run all of them and a `Makefile` to compile them and link them to the model implemented on the home folder.

`CA3` tests These live on `MemDyn_test.cpp`. There are tests for recalling (remember), completing inputs, overlapping (two inputs that share a few bits), random recall, remembering with different strengths, forgetting in general and forgetting a particular entry.

`Hippocampus` tests For the top level component there are a few tests for remembering and forgetting, plus a test of the `random_recall()` method on the file `Hippocampus_test.cpp`

Capacity and scaling tests The implementation tests described on 3.2 are also stored here, on the file `Capacity_test.cpp`. This file contains all the classes used to generate the datasets and run the tests.

References

[1] Beati, Thomas; Carrere, Maxime; Alexandre, Frederic. *Which reinforcing signals in autonomous systems?* Third International Symposium on Biology of Decision Making. 2013

[2] Carrere, Maxime; Alexandre, Frederic, *Émergence de catégories par interaction entre systèmes d'apprentissage*, Conférence Francophone sur l'Apprentissage Automatique (CAP) Philippe Preux and Marc Tommasi eds. 2013

[3] O'Reilly, R. C. and Rudy, J. W. *Conjunctive representations in learning and memory: Principles of cortical and hippocampal function.* Psychological Review, 108(2):311-345. 2001

[4] Joseph LeDoux. *The amygdala* Current Biology, 17(20) :R868–R874. October 2007

[5] J. L. McClelland, B. L. McNaughton, and R. C. O'Reilly. *Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory.* Psychological review, 102(3). July 1995

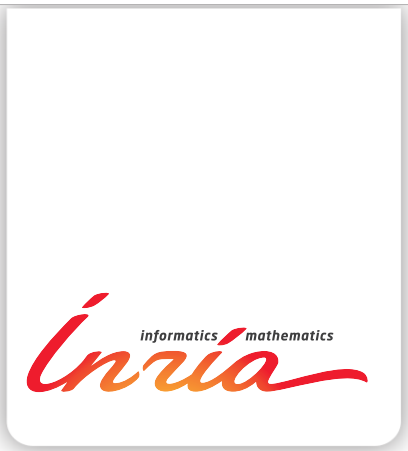
[6] B.L. McNaughton and L. Nadel. *Hebb-marr networks and the neurobiological representation of action in space.* Neuroscience and Connectionist Theory, pages 1–63. Hillsdale, NJ : L. Erlbaum. 1990.

[7] R.A. Rescorla and A.R. Wagner. *A theory of pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement.* Classical Conditioning II : Current Research and Theory, pages 64–99. Appleton Century Crofts. 1972.

[8] N. Schmajuk and J. DiCarlo. *Stimulus configuration, classical conditioning and the hippocampus.* Psychological Review, 99 :268– 305. 1992.

[9] L. R. Squire. *Memory and the hippocampus: a synthesis from findings with rats, monkeys, and humans.* Psychological Review, 99 :195–231. 1992.

[10] Joao Sacramento and Andreas Wichert. *Tree-like hierarchical associative memory structures.* Neural Networks, 24(2) :143–147. 2011.



RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399